PolyLetter

Editorial

Well, this begins the ninth year of PolyLetter. This year I hope to concentrate more on articles about using and adapting the Poly to current needs. I'd like to hear if anyone tries the Parallel printer project in the last issue.

The orientation of many new computer users is to plug in an off-the-shelf software package and go. Unfortunately, there are very few such packages for the Poly. The original Poly users wrote most of their own software. While some of that has found its way into the public domain, it is mostly highly individualized stuff which has to be adapted to the needs of another user.

Frank Stearns has declined to put SPELL 3.0 in the public domain, so if people have been waiting for that possibility, they are out of luck. Frank's answer is essentially "No!, Never!". He states that he continues to offer it at $129.50. I have one second-hand package which I will sell to the first person who comes up with $20, or you can get {A!S} Spell from me for $35.00.

"S-100 Journal", the last periodical devoted exclusivly to the S-100/IEEE-696 bus has finally succumbed to the pressures of the market. Beginning with the Spring 1988 issue, it will be known as "Supermicro" and will incorporate the VMEbus and Multibus in future issues.

LAN for Poly's

Poly Peripherals, a Stone Mountain, GA based provider of Poly-compatible hardware and software products, has just announced a Local Area Network (LAN) for the Poly called the PolyMorphic Integrated Network Environment (PINE).

Proprietor Bob Bybee reports that PINE is the outcome of more than two years of research and development. It is a package of hardware and software which allows 8813 and 8810 systems to share each other's disk files, transparently, over a high-speed network.

Using PINE, users can read and write files on any disk drive in the network. PINE has commands which allow files to be updated without interruption (write locking), to ensure that data is secure.

If you have one Poly with a hard disk, all systems in the network can use it to speed up their disk activity. Disk speed is almost comparable to having a hard disk on each system in the network, even if only one main system has an HD. Floppy or MS-based networks can also benefit from PINE, since it is no longer necessary to make multiple copies of important data files and transport them to each machine. Backups can be done over the network if desired.

PINE uses a standard network interface called ARCNET, which is the same type of network used by IBM-PC and many other systems. Data moves at 2.5 million bits per second, over a distance of hundreds or thousands of feet.

The initial release of PINE contains programs which can perform the disk file-sharing functions described above. Future releases of PINE may include electronic mail and message systems, printer sharing, and... who knows? Your suggestions are always welcome.

Bob reports that Poly Peripherals is committed to supporting users who want to stay with the Poly, and PINE is just the latest in a series of products which enhances and extends the usefulness of PolyMorphic Systems microcomputers.

For more information on PINE contact: Bob Bybee, Poly Peripherals, 5011 Brougham Court, Stone Mountain, GA 30087 (404) 498-3556

Letters

Dear Ralph, December 2, 1987

I'm still having trouble with Poly BASIC version C04. While the Exec/96 addendum states that ON ERROR and ON ESCAPE may be followed by any legal BASIC statement, my copy only supports ON ESCAPE GOTO, as in earlier BASIC versions. Furthermore, the new CALL functions, such as CALL(0), which is supposed to exit to the Exec, get me a "syntax error" message and nothing more. If you have a copy of BASIC C04, and it has the same problem, is there any point in complaining to PolyMorphic Systems? By the way, have you ever disassembled BASIC? Is there any point in doing so? Is there any way to achieve the same result as CALL(0), namely to exit to Exec from within a BASIC program?

You're always soliciting suggestions for features, so here is mine: an introduction to the Poly Exec for the intelligent layman. I bought the System Programmer's Guide some years ago to teach myself the nitty gritty of Exec, but discovered what

I should have already surmised, namely that it is written for an experienced system programmer to acquaint him with the particular features of Exec, which doesn't answer my purpose. It is a reference, not a textbook. The article (actually series of articles) I have in mind would be the counterpart of the many excellent books on CP/M aimed at the intelligent and curious layman, e.g. System Programming under CP/M-80 by Lawrence Hughes. Once you finished serializing the articles in PolyLetter, you could republish them through Abstract Systems on disk or in hardcopy. Ideally, the series would end up including all the information previously published in PolyLetter on modifying sections of Exec for special purposes. I don't ask much, do I?

On another subject, I've designed a neat little gadget to make soft-sectored diskettes into hard-sector ones. A toolmaker friend of mine is making up a prototype. A few years ago, such an appliance would never have justified its cost, but now the price difference between h.s. disks (if you can find them) and the widely discounted soft sector diskes might just make it worth having. If any PolyLetter readers are interested, ask them to contact me. The device can produce 16 or 10 hard sectors by changing an index plate, so if anyone out there is nursing an old Vector Graphic...

Okay, enough for now. Regards, -- F Mark de Piolence, San Diego, CA

P.S. Wanna laugh? See the attached...

[Mark submitted a page from the 1986 DATAPRO index of Minicomputer systems which described the System 8813 as having been first delivered in January 1984 configured as a 16 bit, multitasking system using the iAPX186 CPU operating at 8MHz with 512KB-4MB of main memory and 1.6MB-300MB of disk storage and supporting 16 work stations. Additional dis-information claimed the system as supporting PolyNet and Ethernet, with BASIC, FORTRAN, PASCAL, COBOL and C available for $36,000. -- Is there something Poly Isn't telling us? -- Ok, ok, you can stop rolling in the isles now. This is probably one of Poly's early 'wish list' promotion sheets sent out before the 'new system' was prototyped. It kind of makes one wonder about the validity of information in all those catalogs out there.

In regard to BASIC. It sure doesn't sound like you've got BASIC C04. I have had no problems with mine for the things you describe. I suggest you run CHECKSUM and look at the checksums for the various BASIC overlays and BASIC itself. The checksums should come out as follows. If they don't all agree, then you haven't got a valid copy of BASIC C04.

Size	Cksm	Name
8	9599	Berr.OV
7	809E	Bslv.OV
8	A0	Bfun.OV
8	16F6	Bdir.OV
2	BFF5	Xref.OV
52	DA10	BASIC.GO

If you bought your copy of Exec/96 direct from Poly, and these checksums do not agree on the original purchase disk, Poly should replace it at no charge. The same applies if you got it from a dealer. However, if you aquired Exec/96 with the purchase of a second hand Poly, then the original owner warrenty doesn't apply.

The effect of Z=CALL(0) can be achieved in a couple of ways. First, CLOSE all open files, and then execute a CALL to Warm (HEX 0403 = Decimal 1027) with the code "Z=CALL(1027)". Another way is to put the "BYE" command in the keyboard buffer and then execute the STOP statement, but this does not work properly in command file mode. (OUT 0, "BYE"+CHR$(13) \STOP) Most of my programs exit with: PRINT "Back to ",CALL(1027)

In regard to your feature suggestion, I would need to know more about what you wished to accomplish, and could then solicit or write articles addressing specific needs.

Exec, itself, reads a command line from the user and then scans it to see if it is a command. If so, either executes the command, or more commonly, transfers control to the system overlay which knows how to execute the command. If Exec doesn't recognize a command, then it presumes that the command line contains a file-name to process. If it finds the file, it looks at the extension to see if it is a special case (.BS requires loading BASIC). Otherwise, it looks to see if the file is "runnable" (has a non-zero load address). If the file is not runnable, it presumes that it is a command file and sets up to take input from that file. Otherwise, it returns error code 0300 (I can't find that file) through Err to the error message overlay (Emsg.OV) which prints the message and then goes back to Exec via Warm.

I am offering a used copy of Chuck Thompson's "Addendum to the PolyMorphic Systems System 88 BASIC and User Manuals" which is a "compilation of hidden and obscure features in the PolyMorphic 8810/813 operating system". (See ads).

I still have 10 sector hard disks for sale, as does Al Levy (see the Ads). I also have a supply of 16 hard sector disks if anyone is interested (CALL). But, your device may be needed in the future. How much does it cost? -- Ed.]

Submit.GO

One of the limitations of Poly's command file mode is the inability to use parameters. CP/M had a utility program, SUBMIT.COM, which allowed processing a file of commands, in a file with the extension .SUB. SUBmit files allowed parameters on the command line, which were supplied as answers to inputs requested by the program or command being run. The .SUB file contained a marker to show where the parameters were to be substituted. The marker consisted of the dollar sign '$' together with a numeral to indicate which parameter was to be substituted at that point. '$1' meant substitute the first parameter on the list supplied. Later, MS-DOS provided the same features as built in, but

used the percent sign '%' as the marker.

I do a lot of assembly language work and am frequently executing the following command file (ta.CD):

```
?DE OutPut-file.GO
Asmb <?<Input-file.AS OutPut-file.GO
N
N
N
OutPut-file.GO
```

Well, I got tired of creating the same command file with different names in it every time I began a new assembly language project, and decided to create the equivalent of the SUBMIT.COM program for the Poly. I called the program Submit.GO

Now my 'submit' command file (ta.SB) has the following in it:

```
?DE %2.GO
Asmb <?<%1.AS %2.GO
N
N
N
%2.GO
```

And I can execute it with the command:

Submit ta Input-file OutPut-file

Submit loads itself up under the top of memory and saves a copy of the command line, and begins to watch WHO. Whenever a % is processed from the command file, Submit checks to see if it is followed by a number, and if so, supplies the appropriate parameter instead of getting the input from the command file.

Now, whatever I put in place of Input-file gets substituted wherever the characters '%1' appear in the command file. I don't have to edit or create a new command file for each project. I just submit this one file with the appropriate parameters.

Another place I use this is in searching the PolyLetter mail list for a name. Instead of answering 4 or 5 questions from the program, the answers are all supplied on the command line when I Submit the file.

Of course, if there aren't enough parameters on the command line, then Submit gets the answer from the keyboard by temporarily turning command file mode off, just like SUBMIT.COM. would have.

To obtain a copy of Submit.GO send $15 to Ralph Kenyon.

The Serial Device Driver

by Ralph Kenyon

Readers have asked for an explanation of the serial device driver code. That code is not simple. The serial device is the 8251A USART (Uniform Synchronous Asynchronous Receiver Transmitter) chip on the CPU card. The USART sends data to and from the Poly bus in a parallel mode, but converts the data to or from a serial bit stream for communicating through the RS-232 serial interface.

A meaningful explanation of the code requires a discussion of the organization of the USART and how it is interfaced to the Poly CPU. The USART is organized into a transmitter section and a receiver section. There is an internal transmit buffer and an internal receive buffer. During transmission the USART gets a character from the CPU into the internal transmit buffer. The bits of the character are shifted out one at a time and serially sent out. During receiver operations bits are serially received and shifted into the internal receive buffer. The internal buffers are big enough to hold one character while another is being processed.

When the Poly has a byte to send, it passes that byte to the USART. The USART sends the bits out the serial port one bit at a time. In asynchronous mode it tacks on a start bit and one or more stop bits. It may also add a parity bit.

When the USART receives a stream of bits in the proper format it strips off the start, stop and parity bits and puts the bits into a byte. It then signals the CPU that it has a byte for it.

The CPU can't give bytes (characters) to the USART willy-nilly; it must wait until the USART is free to take a character. For this purpose, the USART has a status word which the CPU can read. The status word is made up of 8 bits; each bit has a particular meaning. The lowest bit, 01H, signals that the transmitter is ready for another character. The next bit, 02H, signals that the receiver has a character ready for the CPU. The next bit, 04H, signals that the Transmitter is empty. This bit differs from the transmitter ready bit in that this bit means only that the transmitter is empty. The 01H bit also means that the transmitter has been enabled by the command word, of which I will say more later. The next bit, 08H, signal that a parity error has occured in an incoming character. The 10H bit signal that an overflow error has occured. This happens if the CPU did not take a character before another was received. 20H means that a framing error occured; that is, that the correct sequence of stop bits was not detected when they were expected. The 40H bit signals that the synch character has been received (for synchronous mode only). The 80H bit signals that the RS-232 signal DSR is good.

```
TxRDY   EQU 1       ;transmitter is ready
RxRDY   EQU 2       ;receiver has a character ready
TxEMPTY EQU 4       ;Transmitter is empty
PE      EQU 8       ;parity error
OE      EQU 10H     ;overflow error
FE      EQU 20H     ;framing error
SYNDET  EQU 40H     ;synch character received
DSR     EQU 80H     ;RS-232 signal DSR
```

To send characters, the CPU must read the status word, wait for the transmitter ready bit to go on and then give the USART the character. Of course, the USART transmitter must have previously been turned on by the CPU.

To receive characters the CPU must read the status word, wait for the receive buffer full bit to go on and then get the character from the USART. Again, the USART receiver must have previously been turned on by the CPU.

The device driver code must get incoming characters and put them in a receive buffer; it must get outgoing characters from a send buffer and give them to the USART. The driver code must handle turning the USART on or off, getting characters from the send buffer and giving them to the USART, and getting characters from the USART and putting them in the receive buffer.

The code for handling the passing of characters to and from the USART is interrupt driven. Signal lines on the USART are connected on the Poly CPU card to one of the interrupt lines. If the USART is turned on, it will generate an interrupt when the transmitter is empty, the received is full, the transmitter is ready or the synch character is detected. The interrupt code must decide which of these conditions caused the interrupt and take appropriate action. Usually it will be that the USART has a character or that it is ready for another. If it is ready for another and there are no more to send then it must be turned off.

When an interrupt is generated the code must read the status word and determine which condition generated it. If the USART has an incoming character the code must get the character and put it in the receive buffer. If the USART is ready for another character, the code must see if there is another character in the send buffer; if so, it must get that character and give it to the USART. Otherwise, the code must turn the transmitter off. Since the USART is only operated in asynchronous mode, the sync detected condition will not occur and need not be programmed for. Transmitter empty and transmitter ready are redundant so no additional code is required.

Lest you get on my case about turning the USART on, that is not done in response to an interrupt. Interrupts are only generated if the USART has already been turned on (one exception I'll mention later). The overall structure of the interrupt code can be illustrated by the following ADA(tm) language fragment.

```
PROCEDURE INTERRUPT_SERVICE_ROUTINE is
BEGIN
 IF READY(TRANSMITTER)
  THEN
   IF NOT EMPTY(SEND_BUFFER)
    THEN
      GET(CHAR,SEND_BUFFER);
      GIVE(CHAR,USART);
   ELSE
     TURN(TRANSMITTER,OFF);
   END IF;
 ELSEIF READY(RECEIVER)
 THEN
   GET(CHAR,USART);
   PUT(CHAR,RECEIVE_BUFFER);
 ELSE
   NULL; -- Spurious interrupt?
 END IF;
END INTERRUPT_SERVICE_ROUTINE;
```

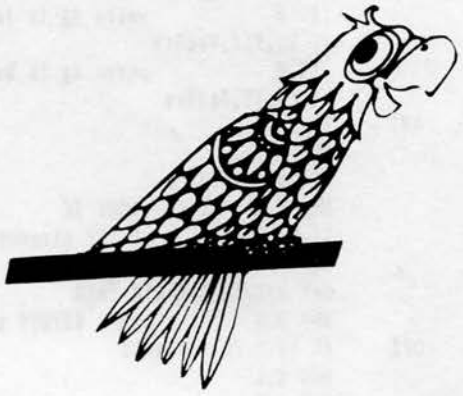
The non-interrupt code is the interface to our programs. That code performs 4 functions. It initializes the device, puts a character in the send buffer, gets a character from the receive buffer, or disconnects the device.

```
TYPE MODE IS (INITIALIZE, OUTPUT, INPUT, DISCONNECT);
FUNCTION : MODE := INITIALIZE;

procedure DRIVER (input => FUNCTION, inout => CHAR) is

BEGIN;
case FUNCTION is
 when INITIALIZE =>
       EMPTY(SEND_BUFFER);
       EMPTY(RECEIVE_BUFFER);
       CONNECT(INTERRUPT_SERVICE_ROUTINE);
       TURN(TRANSMITTER,ON);
       TURN(RECEIVER,ON);
 when OUTPUT     =>
       LOOP
       IF NOT FULL(SEND_BUFFER)
        THEN PUT(CHAR,SEND_BUFFER);
          IF NOT ON(TRANSMITTER)
          THEN TURN(TRANSMITTER,ON)
          END IF;
        EXIT;
        ELSE NULL; -- continue looping until sent
       END IF;
       END LOOP;
 when INPUT      =>
       IF NOT ON(RECEIVER)
       THEN TURN(RECEIVER,ON);
       END IF;
       LOOP
       IF NOT EMPTY(RECEIVE_BUFFER)
       THEN GET(CHAR,RECEIVE_BUFFER);
        EXIT;
       ELSE NULL; -- continue looping until character got
       END IF;
       END LOOP;
 when DISCONNECT =>
       LOOP
       IF EMPTY(SEND_BUFFER)
        THEN EXIT;
       ELSE
          IF NOT ON(TRANSMITTER)
          THEN TURN(TRANSMITTER,ON)
          END IF; -- continue looping until sent
       END IF;
       END LOOP;
       TURN(TRANSMITTER,OFF);
       TURN(RECEIVER,OFF);
       DISCONNECT(INTERRUPT_SERVICE_ROUTINE);
  END DRIVER;
```

Here is a disassembly of the actual code for the serial driver with conditional assembly points for some of the various versions -- Printer/37, Printer/40, and Printer/42. This code also does a few more things. It allows for a buffer limited type device, as well as for inserting pad characters after various ASCII characters.

```
;Program name Sio.PS
;Disassembled on 12/6/80
```

```
;Serial Input /Output driver
        REFS SYSTEM.SY
        REF     ;Read in ALL system labels.

        REFS ASCII
        REF     ;Read in ALL ASCII equates.

V37     SET FALSE       ;Set for Printer/37
V40     SET FALSE       ;Set for Printer/40
V42     SET TRUE        ;Set for Printer/42

UISR    EQU SRA4        ;These aren't in SYSTEM.SY
USRTSR  EQU 054H        ;Nor this one.

;       PORTS

DATA    EQU 0
STATUS  EQU 1
CONTROL EQU 1

;       STATUS

TxRDY   EQU 1           ;transmitter is ready
RxRDY   EQU 2           ;receiver has a character ready
TxEMPTY EQU 4           ;Transmitter is empty
PE      EQU 8           ;parity error
OE      EQU 10H         ;overflow error
FE      EQU 20H         ;framing error
SYNDET  EQU 40H         ;synch character received
DSR     EQU 80H         ;RS-232 signal DSR

;       COMMAND

TxEN    EQU 1
DTR     EQU 2
RxE     EQU 4
SBREAK  EQU 8
ER      EQU 10H
RTS     EQU 20H
IR      EQU 40H
EH      EQU 80H

;       Some Macros

Set     MACRO
 IF NULL[#1]
#L      XRA A
 ELSE
#L      MVI A,#1
 ENDIF
        OUT CONTROL
        ENDM

SuBu    MACRO
#L      LXI D,#1
        MVI B,#2
        LHLD #3
        CALL #4
        ENDM

set     MACRO   ;Sets #1 to the value of #2, 0 otherwise.
 IF NULL[#2]
#L      XRA A
 ELSE
#L      MVI A,#2
 ENDIF
        STA #1
        ENDM
```



```
JEQ     MACRO   ;JUMP IF EQUAL
#L      JZ #1
        ENDM

REQ     MACRO   ;RETURN IF EQUAL
#L      RZ
        ENDM

JGE     MACRO   ;JUMP IF GREATER OR EQUAL
#L      JNC #1
        ENDM

RGE     MACRO   ;RETURN IF GREATER OR EQUAL
#L      RNC
        ENDM

JLT     MACRO   ;JUMP IF LESS THAN
#L      JC #1
        ENDM

RLT     MACRO   ;RETURN IF LESS THAN
#L      RC
        ENDM

JNE     MACRO   ;JUMP IF NOT EQUAL
#L      JNZ #1
        ENDM

RNE     MACRO   ;RETURN IF NOT EQUAL
#L      RNZ
        ENDM

If      MACRO   If #2 is #1 then GO to #3
#L
 IF LEN['#2']=1
  IF '#2'='A' OR '#2'='B' OR '#2'='C' OR '#2'='D' OR '#2'='E' OR
'#2'='H' OR '#2'='L' OR '#2'='M'
        CMP #2  ;#2 is a register A B C D E H L or M
  ELSE
   IF #2=0
        ORA A   ;#2 is 0
   ELSE
        CPI #2  ;#2 is immediate data or label
   ENDIF
  ENDIF
 ELSE
  IF #2=0
        ORA A
  ELSE
        CPI #2
  ENDIF
 ENDIF
 IF NULL[#3]
        R#1     ;#1 is one of EQ, NE, LT, or GE
 ELSE
        J#1 #3
 ENDIF
        ENDM

ck      MACRO   ;check if #1 is #2 to #3 then goto #4
#L      LDA #1
        If #2,#3,#4
        ENDM

        ORG 03000H      ;Device drivers live here
        IDNT $,$        ;Load and start addresses

        DCR B           ;B contains function
        JZ Putc         ;1 = output char to device
```

```
        DCR B           ;2 = input char from device
        JZ Getc         ;3005 CA4831
        DCR B           ;3 = initialize device
        JZ Init
        DCR B           ;4 = disconnect device
        JZ Kill         ;300D CA3F30
        STC
        RET

Init    LXI H,0         ;Initialize device
        SHLD RBPtrs     ;3015 229431
        SHLD TBPtrs     ;empty buffers
        set RCount      ;clear count
        STA achflg      ;clear flag
        LDA speed       ;set up baud rate
        STA BdRate      ;3025 322B30
        CALL SETUP
BdRate  DS 1            ;Place for baud Rate
        DB 10101010B    ;Times 16, 7 bits, disabled
                        ;even parity, 1-1/2 stop bits
        DB 01000000B    ;Internal reset to mode
                        ;instruction format
        DB 11011010B    ;Times 16, 7 bits, enable
                        ;odd parity, 2 stop bits
        DB 0            ;all functions disabled
        LXI H,ISR
        SHLD UISR       ;set up interrupt service routine
        Set RTS+RxE+DTR ;turn on receiver, DTR, & RTS
        set USTATS      ;mark the usart free
        RET

 IF V37 OR V40
Kill    Set IR          ;Reset to mode
        LXI H,USRTSR
 ENDIF
 IF V42
Kill    Set ER          ;Reset errors
        Set
        LXI H,loret
 ENDIF
        SHLD UISR       ;clear interrupt service routine
        RET

;This is the actual interrupt service routine
ISR     LXI H,loret     ;304AH
        PUSH H
        IN STATUS       ;Ok, who dat? Whaddya want?
        RAR
        JC TXRDY        ;Go send another one
        RAR
        RNC             ;Spurious interrupt
        IN DATA         ;Got one coming in
        ANI 7FH
        MOV C,A
        ck ackchr,NE,C,StowCh
        set achflg      ;Reset achflg
        ck blim,NE,0    ;fall thru if blim=0
StowCh  SuBu RBuff,RBSize,RBPtrs,PutCh
        RZ              ;Lose char if buff full!
        SHLD RBPtrs
        RET

TXRDY           ;Transmitter ready for another one
 IF V40 OR V42
        ANI DSR SHR 1   ;Status is right shifted
        JZ NoDSR        ;307D CAA830
 IF V40
TXRDY1
 ENDIF
 ENDIF
        SuBu TBuff,TBSize,TBPtrs,GetCh
        JZ NMTS         ;Oops.. no more to send
        SHLD TBPtrs
        OUT DATA        ;send it
        RET

NMTS    IN STATUS       ;No more to send
        ANI TxEMPTY     ;3096 E604
        JZ NMTS         ;Loop to wait until it's sent
        Set RTS+ER+RxE+DTR
        set USTATS      ;then turn transmitter off
        OUT DATA        ;Stuff 00's into the buffer to shut
        OUT DATA        ;off transmitter empty interrupt.
        RET             ;30A7 C9

 IF V40 OR V42          ;Gotta have DSR up for these versions
NoDSR   CALL NMTS
NoDSRw  EI
        HLT
        IN STATUS
        RAL             ;30AF 17
        JNC NoDSRw      ;Go wait for it to come up
        DI
        Set RTS+RxE+DTR+TxEN
        STA USTATS
 IF V40
        JMP TXRDY1
 ENDIF
 IF V42
        RET
 ENDIF
 ENDIF

Putc    LXI H,padchr
        INX H           ;move up to crpad
        If EQ,CR,PadRtn
        INX H           ;move up to lfpad
        If EQ,LF,PadRtn
        INX H           ;move up to tabpad
        If EQ,TAB,PadRtn
        INX H           ;move up to bspad
        If EQ,BS,PadRtn
OP1     ANI 7FH
        MOV C,A
        LXI H,USTATS
        MOV A,M         ;30DF 7E
        If NE,0,OP2     ;USART already turned on
        DI
        Set RTS+ER+RxE+DTR+TxEN
        MOV M,A         ;(Set USTATS non-zero)
OP2     ck blim,EQ,0,SendCh
        MOV E,A
        MOV A,C
        CPI ESC
        MVI B,1
        JNZ $+5
        MVI B,3
        ck RCount,GE,B,SendCh   ;RCount >= B
        EI
        ;
OPW     ck achflg,NE,0,OPW      ;Loop waiting for buffer empty
        MOV A,E
        RAR
        STA RCount      ;blim/2 starts count
        PUSH B
        set achflg,-1   ;set achflg
        LDA etxchr
        MOV C,A
```

```
        CALL SendCh
        LDA stxflg      ;311D 3AE731
        ORA A
        LDA stxchr
        MOV C,A
        CNZ SendCh      ;3125 C42931
        POP B
SendCh  LXI D,TBuff
        MVI B,TBSize
        DI
        LHLD TBPtrs
        CALL PutCh
        JZ SChWait      ;3135 CA4131
        SHLD TBPtrs
 IF V37
        EI
        LXI H,RCount
        DCR M           ;count down
 ENDIF
 IF V40 OR V42
        LXI H,RCount
        DCR M
        EI
 ENDIF
        RET

SChWait EI
        HLT
        JMP SendCh

GetCw   EI              ;Getc-2
        HLT             ;3147 76
Getc    LXI D,RBuff
        MVI B,RBSize
        DI
        LHLD RBPtrs
        CALL GetCh
        JZ GetCw
        SHLD RBPtrs
        EI
        RET

PadRtn  MOV B,M         ;Get pad count
        INR B           ;decrement first
        PUSH B          ;save remaining count
        CALL OP1        ;put out the char
        LDA padchr      ;get the pad char
        POP B           ;get count
        DCR B           ;see if more to send
        JNZ PadRtn+2    ;yup, so pad it some more
        RET             ;else we're done

PutCh   MOV A,H         ;Get char pos in A
        CALL PtoCh      ;and move up to char
        MOV A,C         ;Get char
        STAX D          ;Put char in buffer
        INR H
        MOV A,H
        If NE,B,$+5
        MVI H,0
        MOV A,H
        CMP L
        RET

GetCh   MOV A,L         ;Get char pos in A
        If EQ,H         ;Compare to char pos - Same so none
        CALL PtoCh      ;Point to th char now
        INR L
        MOV A,L
        If NE,B,$+5
        MVI L,0         ;need no roll position pointer
        ORA B
        LDAX D
        RET

PtoCh   ADD E           ;add in buffer addr
        MOV E,A         ;to get char position
        RNC             ;in DE
        INR D           ;Got to roll upper
        RET             ;byte too

TBPtrs  DS 2    ;Transmission Buffer pointers
RBPtrs  DS 2    ;Receive Buffer pointers
RCount  DS 1    ;Receive Count
achflg  DS 1    ;Acknowledge character

RBuff   EQU $   ;Receive buffer starts here
RBSize  EQU 5
TBuff   EQU RBuff+RBSize        ;Transmission buffer here
TBSize  EQU DDbase-TBuff

        ORG DDbase

        END
```

Public Domain

This issue I have an original games demo disk from PolyMorphic Systems. PGL-V-26 has Exec/4D, and BASIC A01. Some of these programs have been available, but this is their original form as distributed by Poly back in 1977.

```
Disk PGL-V-26 has 29 files on it, 32 free entries.
338 sectors in use, 0 sectors deleted, 12 sectors free.
 Size Name.       Size Name.    Size Name.
   20 HAMURABI.BS   24 CRAPS.BS   16 HANGMAN.BS
   13 REVERSE.BS    14 HORSE.BS   22 BACKGAMMON.BS
   20 STOCK.BS       7 SHOOT.BS    9 MATHDRILL.BS
   19 PLOT-DEMO.BS  22 STARTREK.BS 1 PLOT.BS
   21 LANDER.BS
```

HAMURABI.BS - As the ruler of ancient Samaria, you are responsible for the management of a kingdom.
CRAPS.BS - Place your bets.
HANGMAN.BS - Can you guess his word?
REVERSE.BS - See how quickly you can get the numbers in order.
HORSE.BS - Poly Graphics demonstration with 'racing' horses.
BACKGAMMON.BS - Poly acts as the referee for two players.
STOCK.BS - Try a simple stock market simulation.
SHOOT.BS - A simple graphics arcade game of clay pigeons.
MATHDRILL.BS - How good are you at multiplication?
PLOT-DEMO.BS - Demonstrates graphics plotting.
STARTREK.BS - An early (the earliest?) version.
PLOT.BS - Plotting a simple function.
LANDER.BS - Try to land on the platform.

Readers Responses

Try to change the tone to the present and expectant future. When I read PolyLetter I get a feeling of "once was"? It's living in the past! -- John Matelock, Cambridge Springs, PA.

[While the Poly is a computer of the Past, it still has a future with us. -- Ed.]

I'm glad to see the tradition has been carried on from generation to generation. You're doing a fine job in much greated depth that ever before. Sure brings back memories of the days when I started the PolyLetter ... that was when REAL men had REAL computers! -- Mark Sutherland, Norcross, GA.

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

OUT0, OUT0, DAMNED SPOT

by Norm Shimmel

BASIC C03 and, I believe, C04 have a bug in them that is driving me up the wall. If you have a program that uses the OUT 0, function in BASIC, you will find difficulty using the printer after you run your program. For instance, if your program has the line:

```
100 OUT 0,"LIST"+CHR$(13)\STOP
```

which will stop the program and list it by putting the LIST command and a carriage return in the keyboard buffer, it will work fine. But when you

exit the program and try to FORMAT a text file to your printer, you may find a very strange message awaits you.

Ralph tells me that wormhole 5 is used by the OUT 0 function but is not restored to normal when it's done. This causes complications since printer routines also use WH5.

To correct this problem, reinstall the printer driver either from BASIC with

```
XXXX Z=CALL("Prnt",1)
```

or from EXEC with

```
Printer [printer name]
```

That should clear up your problem.

[Because Norm submitted this, I am following it up with the associated Abstract Systems BugNote (out of sequence) which contains additional information. I will resume the sequence with the next issue. Ed.]

BUGNOTES

Abstract Systems BugNote 024.0 May 26, 1983

BASIC C03, 04/14/81 has a bug in the OUT command. The OUT command takes over Wormhole 5 to process characters to the destination device. (OUT 0,stuff places stuff in the keyboard buffer, OUT 1,stuff sends stuff to the display, 2 or 3 sends stuff to the printer or custom device. 4 or more sends stuff to a disk file channel.)

The bug is that WH5 is NOT restored after the OUT command processing is completed. WH5 is left with garbage in it. Trying to use FORMAT.CO after running a BASIC program which used the OUT command may have unpredictable results, because WH5 is left pointing into the Bfun overlay, user memory, or somewhere else. Any other program which uses WH5 will have similar results. MAILIST.BS and other programs use the OUT command, so use of FORMAT.CO following MAILIST.BS may bomb.

This is a potentially serious bug in that it could result in spurious writing on disk! I discovered the bug by getting Error 0101 - Dio says: Bad parameters! If the disk address had been in range, an unknown number of sectors of garbage could have been written anywhere on disk, producing a catastrophic crash.

To avoid this bug, WH5 must be restored after using any BASIC program which uses the OUT command. WH5 is restored by the system command, Printer name, for any printer which uses the Sio.PS driver. Simply re-initializing your printer will restore WH5 and avoid this bug.

HELP!

In this section I share with you the help system files I have built up over the last few years. (The entire system is included with Abstract Systems Exec.)

```
$HELP COMMAND Sniff

        HELP file for system command "Sniff"

The "Sniff" command checks a disk for errors by reading
one sector at a time, starting with the last used sector.

Syntax: "Sniff [n sss]" (RETURN)
'n' is a drive number and 'sss' is the starting sector.

"Sniff" checks the disk in the system resident drive.
"Sniff [n]" checks all used sectors on drive 'n'.
"Sniff [n sss]" starts with sector 'sss'.

Minimum size: "Sn"      Examples: "Sn 2", "Sn 1 1F0"

(Note: Poly's Exec has no 'sss' feature.)
```

How to get your PC to PAGE.

The PRINT command automatically outputs a form-feed after printing a file, so all we need to do is to create a file which has no characters in it. PRINTing that file will result in only a form-feed being sent to the printer.

First, use EDLIN to create file called PAGE.TXT. Since EDLIN won't create an empty file, this must be done in two steps.

```
EDLIN PAGE.TXT
I
F6 (CTRL-Z)
E
EDLIN PAGE.TXT
1,#D
E
```

Second, use EDLIN to create a file called PAGE.BAT.

```
EDLIN PAGE.BAT
I
PRINT /D:PRN PAGE.TXT
F6 (CTRL-Z)
E
```

Readers Requests

HELP! I have double sided REMEX 8" drives for my MS, but I don't know how to strap the shunts. Anyone who knows how to configure these drives for the Poly MS please write Karl Thomas, 145 Bond Street, Elk Grove, IL, 60007-1218

Other readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. What happens when a BASIC program is saved with SAVEF, or SAVEP. How to UNSAVEP. How would CP/M be of use. An explanation of the Front Panel. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Hardware update recommendations, Source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer

all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

BASIC

Here's one trick to save space in your BASIC program when initializing string variables. A$="" requires 5 bytes of program space, so the brute force method where one simply sets the string to the literal string containing the right amount of blanks, takes 5 bytes plus the number of blanks. The following code takes only 21 bytes regardless of how big A$ is (K=9 goes up to 511 bytes).

```
FORI=1TOK\A$=" "+A$+A$\NEXT
```

For any string longer than 16 bytes, it takes less program memory space to use the loop method. Additional variables in the loop only require 13 bytes, so the loop is cheaper for two variables of lengths greater than 29 bytes.

Bit Bucket

Thanks to the ACNJ newsletter, we now know that a BYTE is an octobit.

In This Issue

Coming Soon

Using Your IBM Color Monitor with A Poly and a Clone. Modems and Communications software, More BASIC for Beginners, How to UNSAVEP protected Programs, More System Programmers Notes, Converting Poly BASIC to PC BASIC, More Help, BugNotes, Public Domain Software, etc.

Questions

Can you find and answer the questions asked in this issue? Send your answers and requests in.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL

Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter; please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter* are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Overseas $20.00 yr., payable in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).

PolyLetter

Editorial

How many programming languages are there for the Poly? By now I have been exposed to several. The first, and most common are "BASIC" (Beginners All Purpose Symbolic Instruction Code) and Assembly language. BASIC was designed for teaching simple programming concepts. Assembly language is designed to facilitate programming in machine language. It gives symbolic names for machine language instructions, and provides some simple house-keeping services.

Tiny-BASIC is a small, but fast version of BASIC. Poly's Tiny-BASIC (TBASIC) has the SPEECH command built in for use with Speech Lab.

Another language is "FORTRAN" (Formula Translation). FORTRAN was designed as a language for use in scientific calculations. It does math very well. PolyGlot Library Volume number 7 (PGL-V-07) has a FORTRAN compiler on it.

Another is "C". I'm not sure of C, but I belive it was designed to provide both the low level acces to the machine that assembly language does, as well as the structured features of high level languages. PGL-V-11 has a SMALL-C compiler on it.

I have also implemented FORTH, which stands for "fourth generation". FORTH is stack oriented and has the features of an operating system and a compiler as well as those of a language.

Another language which has been implemented is PILOT, which is a language designed for computer aided instruction. The kinds of things it does well is display text and match answers. Mendocino Microcomputers wrote SUPERPILOT for the Poly. I have also adapted the North Star Users Group version of PILOT to the Poly.

I have heard that someone once wrote a LISP interpreter for the Poly, but I have never seen it; I have also been working on my own, but have not completed it. LISP is especially good at processing lists of symbols; a sentence is such a list, and LISP is the language of choice for artificial intelligence research.

Yet another language which was worked on, but never released, is PASCAL-MT. PASCAL was designed by Niklaus Wirth for the express purpose of teaching structured programming.

Finally, there's Little-Ada. Little-Ada was first described in October 1979, and developed for the Poly by 1980. ADA™ was designed for Military imbedded systems as a standard language to save costs. "Little-Ada" is a tiny subset of ADA.

As of this issue, I have decided to release "Little-Ada" to the public domain. (See "In the Public Domain" for more information.)

"Ada" is a registered trade-mark of the US DOD to refer to the Ada language. "Little-Ada" was first used by Robert Mathis in documentation prepared in October 1979. Use of "Little-Ada" in reference to this subset pre-dates the DOD trademarking of "Ada" by several months.

Using Your IBM Color Monitor with a Poly and a Clone by Don Haywood

I think that I have heard a cry from the readers of PolyLetter for some material on using a clone. And, unless I read the signs wrong, we still have some hardware hackers out there. When I first bought my Poly it consisted of just the video board and CPU, SD and Godbout memory and a surplus mini-computer power supply. I built the keyboard and enclosure. Eventually, I wire-wrapped a version of the Poly disk controller for myself. In between I built voice synthesis and music into the little bugger.

Well, I like the Poly a lot (it's familiar) and I also own a clone. To me, it's ridiculous to own a nice RGB color monitor and not be able to use it with the Poly. Besides, if done right, why not use the same monitor for both the clone and the Poly? Think of the space you can save by getting rid of one monitor. Keep reading because if you follow these instructions you will be able to toss out the Poly monitor and look at a beautiful Poly chrome display. It's real easy, too.

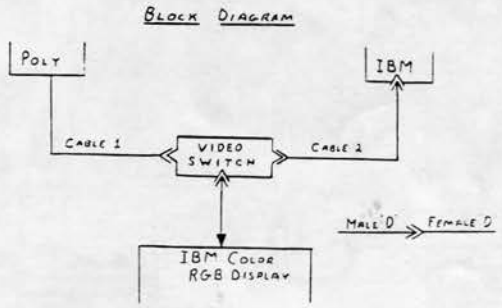


FIG 1

The Video Switch

A block diagram for the video switch is shown in figure 1. As you can see, outputs from both the Poly and clone go into it. One cable leaves and carries video info to the IBM color display. This information can be either Poly or clone stuff. A selector switch determines which computer is hooked up to the display.

Many of you probably already know this but I'll mention it anyway. The Poly sends its' video information out in composite form while the clone uses RGB. To make the switch work, the Poly video must be picked off of the VTI card before it is combined together. Once this is done, the switch is simply a switch.

Getting the Right Video Signals From Poly VTI

The signals which the IBM style monitor requires are: ground, intensity, horizontal drive, vertical drive and three video signals - red, green and blue.

Available on the Poly VTI card are: ground, horizontal drive, vertical drive and one video signal. These are sourced at IC3 and IC31. I have reproduced the relevant parts of the VTI parts placement diagrams and schematics - the ICs are circled on figure 2, the parts placement diagram.

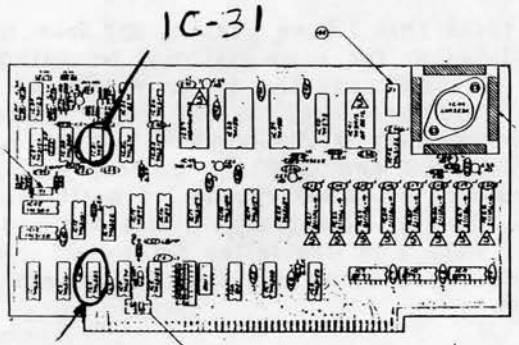
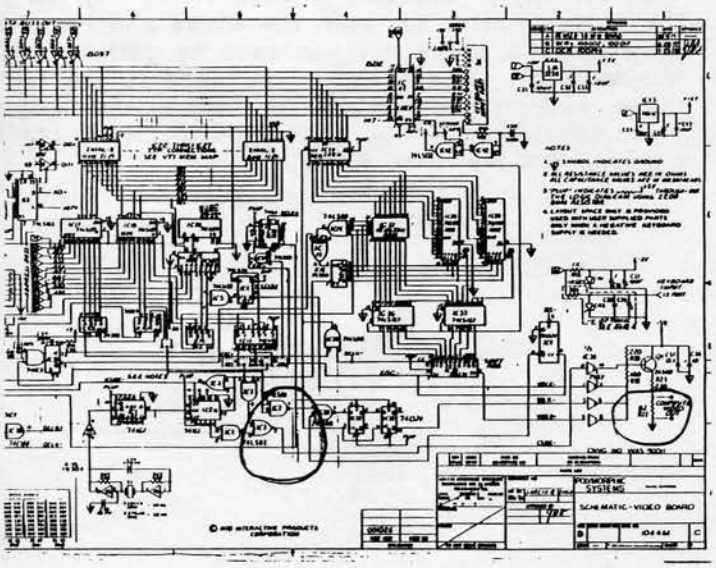
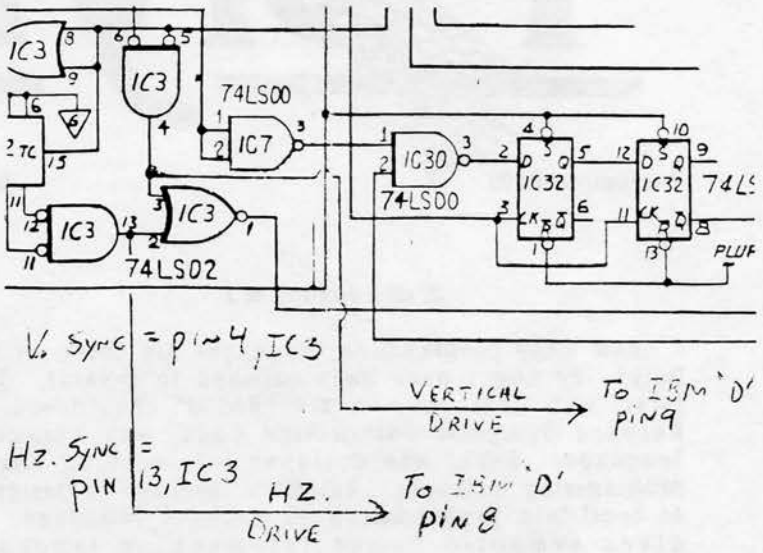
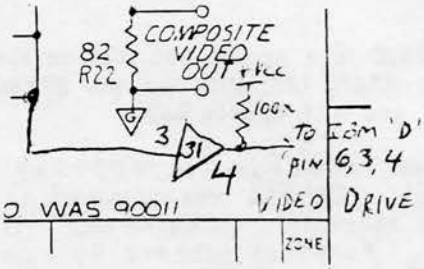


FIG. 2



The points where the appropriate signals are picked off of the ICs are shown on the circled parts of figure 3, the schematic diagram. [Blow-ups show the alterations. -- Ed.]





Please note: IC31 contains one unused driver. I chose to use it as shown in figure 4.

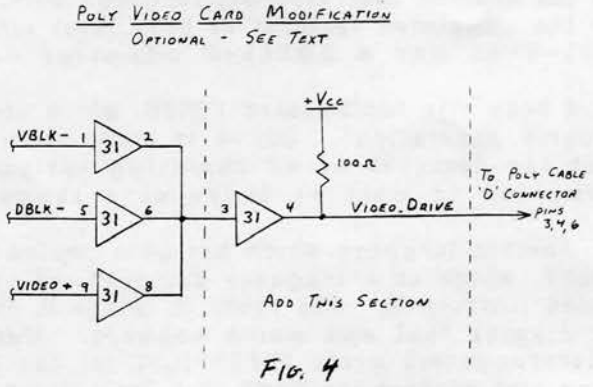


FIG. 4

If you don't want to go through the bother of adding it then take the video drive directly off of IC31, pins 2, 6, or 8.

You must assemble a cable which will interface to the video switch or the IBM display cable. Incidentally, all you really need do is build this cable and simply plug it into your original IBM display cable to use the IBM monitor. I like being able to leave both the clone and Poly hooked up, though.

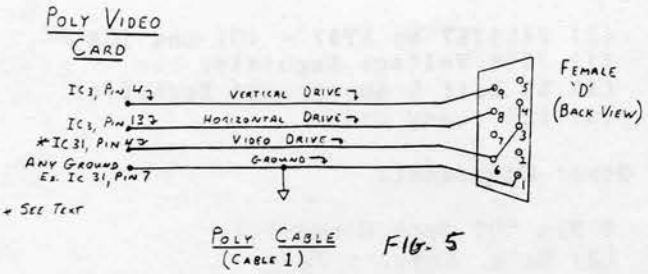


FIG. 5

Figure 5 is a diagram of the new cable - cable 1. It requires four conductors terminated by a standard 9 pin female "D" type socket. I used intercom cable but you can use two speaker wires or four single wires tied together or four of the twenty-five wires in a RS232 cable - you get the idea. Solder one end of each of the four chosen wires directly to the back of the appropriate IC as indicated but please take the IC out of its' socket first. Ground can be pulled from any ground on the VTI board. Make the cable plenty long, six feet at least, to give you enough room to put your Poly anywhere you want.

At this point I should mention that now is the time to choose how you'd like the Poly display to look on the IBM monitor. Figure 6 is a diagram of some of the possible configurations.

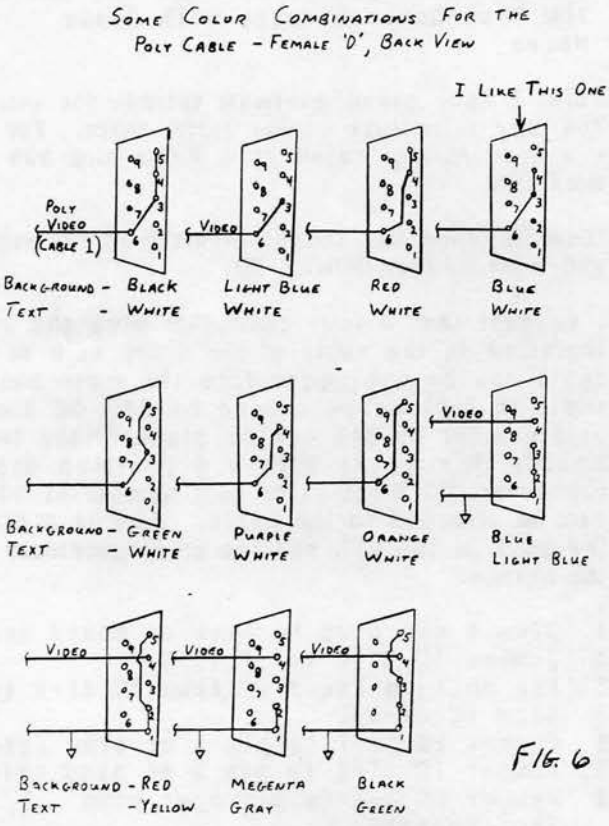


FIG. 6

If you are real boring you'll choose black and white. I really like a blue background with white or pale blue letters. I show the cable hooked up as I like it but you can experiment around.

Another Cable

If you've decided to go whole hog and not fiddle with switching cables then you must build a second cable - cable two. This cable configuration is shown in figure 7.

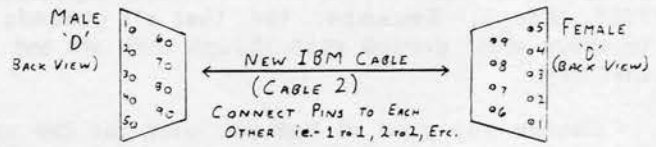


FIG. 7

You'll need to connect all but pin 7. One end is a male 9 pin "D" and the other a female. Once again, make it plenty long to allow you to place your clone in a variety of locations.

The Video Switch Interface

Now probably comes the scariest part outside of actually trying out the completed switch. See figure 8. I used a wire-wrap board and sockets to build the project.

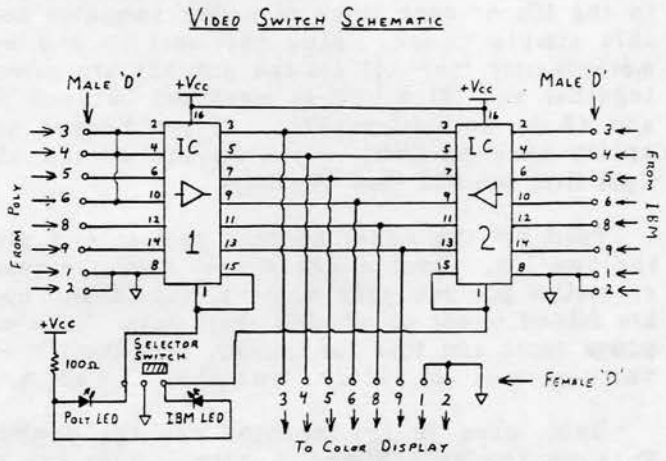


FIG. 8

The actual switching is done by the two ICs. These are nothing more than tri-stated bus drivers and are real cheap. Essentially, the two cables - one from the Poly and the other from the clone are input to separate ICs. The outputs of the ICs are tied together and they go to the RGB display. A selector switch enables or disables the output of each IC. I set the switch up to light a pretty colored LED according to the chosen input. That way, I need not look at the monitor to see what computer is hooked up (so what - I like lights and buzzers!).

You must also wire up a simple 5 volt power supply. See figure 9.

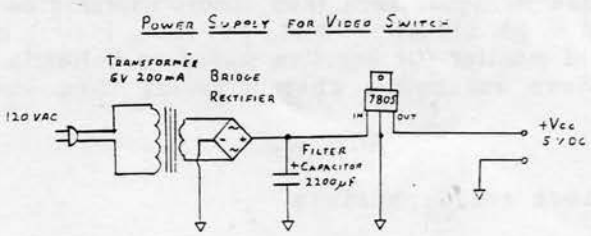


FIG 9.

I put it on the same board as the two ICs but you

can use a 9 volt DC wall adaptor in place of the transformer and bridge if you want to take the sissy's way out. Leave in the filter capacitor and 7805, though. Remember, too, that all grounds hook to every other ground even though they are not drawn that way.

Choose any kind of box you wish for the switch. Just keep in mind that there must be room for the three "D" connectors and, maybe, a transformer inside.

The parts list follows this text.

Final Testing

NOTE: Anything which appears unusual - like smoke - indicates a problem. Turn off right away! [Electronics people call this the "Smoke test" -- Turn it on and see if it smokes. -- Ed.]

After you have it all wired up and before you put in the ICs or hook it up to either computer perform this simple check. Plug the unit in and with a meter verify that: (1) all the grounds are commoned together and (2) 5 VDC is measured between pins 8 and 16 on both IC sockets. If you hooked up the spiffy looking LEDs, clicking the switch should light first one LED then the other.

Turn off the power to your project and plug in the two ICs. Double check that they are inserted correctly; pin one goes where it should and no pins are folded under or missing their hole. Turn on the power again and look for smoke. Feel the ICs - they should get a little warm but not hot.

Next, plug in the monitor and the computers. Turn on the switch and monitor. Flip the video selector switch. In the Poly position the monitor should produce whatever color you decided the background should be - probably blue.

Turn on the clone. Flipping the switch should make the display show IBM stuff in one position and Poly color in the other. If everything seems to be going OK this far turn on the Poly. Now, one position of the selector will show Poly things, too.

Final Comments

Any questions? Forward them to me and I'll try to get back to you. By the way, did you know that the disk drives in your clone can be used for the Poly, too? But that's another story. You can also replace nearly every IC in your Poly with CMOS stuff and make it higher tech then almost anything on the market - no kidding. And, yes Ken, I still have your old monitor ICs but I've moved to Colorado and still have not had a chance to try them out.

Parts List

Resistors and Capacitors

(2) 100 Ohm 1/4 Watt Resistors
(1) 2200 mF 15 Volt Capacitor

Semiconductors

(2) 74LS367 or 8T97 - IC1 and IC2
(1) 7805 Voltage Regulator
(1) 50 Volt 1 Amp Bridge Rectifier
(2) LED - any color

Other Components

9 Pin "D" Type Connectors
(2) Male, Chassis Type
(1) Male, Cable Type (with hoods and strain relief)
(1) Female, Chassis Type
(2) Female, Cable Type (with hoods and strain relief)

(1) 6 Volt, 200 mA Power Transformer
(1) SPDT Toggle or Slide Switch
(1) AC Power Cord

Hardware

(1) Suitable Enclosure
(2) 16 Pin Wire Wrap Sockets
(1) Perforated Breadboard

Miscellaneous

Enough Hook-up Wire for the Two New Cables
IBM Type Color Display with Clone
Nerve

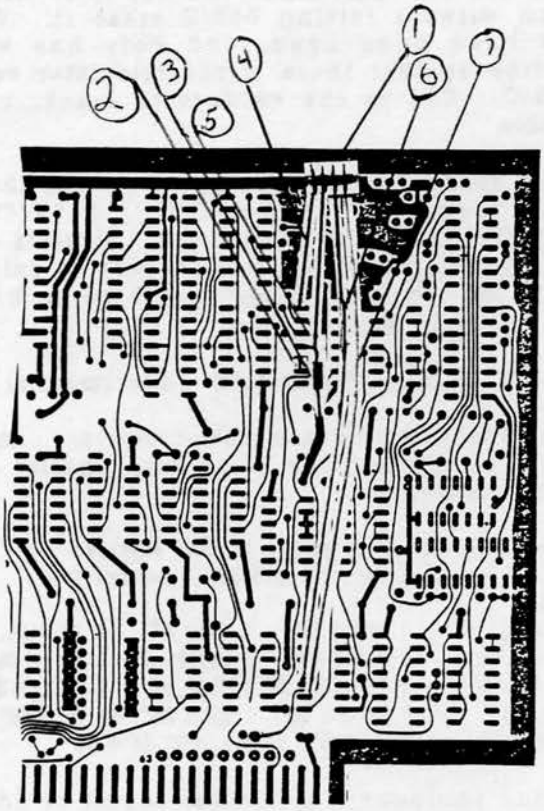
Note: I have given minimum ratings for components. You may substitute higher rated parts. For Example - a transformer rated at 6 Volts and 500 mA will work fine.

[Don Haywood has taught electronics for eight years and lives in Fort Collins, CO.

I suggest that a four conductor plug and socket be installed in the cable at the video card so that the cable can be unplugged from the video board. A 4 position locking polarizing header, GC Electronics part number 41-044 can be glued to the top of the board; this mates with a 4 position dip solder connector, GC Electronics part number 41-004, which can be attached to the cable. Here is a diagram of the back of the VTI, and the steps necessary to make the change.

1. Glue 4 pin plug to back of board near Q-1.
2. Jumper IC 31-2 to IC 31-3.
3. Add pull-up resistor from IC 31-4 to trace from IC 31-14.
4. Jumper IC 31-7 to pin 1 of plug (ground).
5. Jumper IC 31-4 to pin 2 of plug (video).
6. Jumper IC 3-3 to pin 3 of plug (horizontal).
7. Jumper IC 3-2 to pin 4 of plug (vertical).

Of course, the cable must match the pins on the plug. -- Ed.]



CRASH RECOVER

by Ralph Kenyon

I use my VOL program on boot up to tell me what disks I have in each drive. Recently when I booted, it told me:

(Error 0103) is in drive 3.

I looked at the disk and to my horror discovered that it was my (A:S) assembly source disk. SuperZap revealed that sectors 0 and 2 in the main directory were partially trashed.

Sector 2 looked okay, but when I tried to re-write it out SuperZap gave me 0103 back. What? Oh ya, it's in the directory, so it's trying to read sector 0 first. Damn... I can't write sector 2 because it wants to read sector 0 first. Likewise, It won't write sector 0, because it wants to read sector 2 first. Talk about catch 22...

Well, sez me, let's use RECOVER.GO to read the good sectors and put them together with the bad ones. Wait, I've an even better idea. I'll use RECOVER.GO to read all four sectors in the directory, but fix it so as not to croak on a checksum error. First, I load rdb (my version of Poly's RDB.GO, the relocatable debugger program.) Then, I GET RECOVER.GO. Next, I bring up rdb and use the S(earch) function to look for a call to Dio. I put rdb in the instruction mode and press "S" to tell it to look for "CD 06 04" starting at 3200. I look at each call to look for the call to Dio which is a READ, and set up a breakpoint just after the call. Next, I go to Exec and use the START command to start RECOVER.

I tell it to start at sector 0, and end at sector 3, and to give the file a 101 start and load address, and give the file the name root.DX. At this point it execute the call to Dio and returns to the breakpoint. I read that the carry flag is set, and that DE contains 0103 (checksum error). So, I blithely change the FLAG byte to reset the carry flag, so that the program thinks that no error occured.

The program writes out the sector, and reads the next. This time, since the CARRY flag is not set, I just press 'G' and it writes out the sector. One more read gives me sector 2 with the carry set again. This time I clear the CARRY flag and proceed. It writes sector 2 and goes on to read sector 3. One more 'G' and the program terminates normally, giving me a file of four sectors, made up of two good ones and the partial data from two bad ones.

Now comes the tedious part. I edit sectors 0 and 2 (the bad ones) using Szap to correct errors. This requires frequent referal to the actual disk to find the start and end of the files so that the correct directory entry can be made.

Now, to test the validity of the entered data, I call up Printer File, which is my Print to a file utility, so that I can capture the printer output in a file. Next, I set the Printer LOG command so that anything on the screen goes to the printer (in this case to the file). Now, I run my slist.GO program which gives a sorted listing of the entire disk in one directory format. Of course, the output on the screen is also going to the file. After 'slist'ing the disk, I close the output file and edit it to clean out the Directory header information, leaving me with a pure list of files in the ENABLEd format. Next, I use SORT.GO with the column selected at the Disk Address location to put the files in the order that they appear on disk (regardless of subdirectory).

Now, to test the validity of this listing, I create a BASIC program which will read the list item by item and add the file size to the disk address to compute the disk address of the next file. It tells me if the disk address of the next file doesn't match the disk address of the preceding file plus its size. If not, it's back to Szap, check out the disk and correct the directory entry before running the whole shebange again. When I Finally get a successful run thru, it's time to backup the disk!

First, I use SuperZap to Zap the directory, and then enter a dummy number of used sectors to say the disk is completely full. Next, I IMAG the disk to get a copy of the data even though the directory is gone. Next, I use my Rebuild.GO program to enter a file name in the directory which has 4 sectors, start and load address of 101, and a DX extension, but I say the first disk address is 0, which is where the root directory would be. Once this is done, I use Scopy to copy the reconstructed directory, root.DX, onto this new file.

Scopy tries to do the job, but errors out with I can't find that file. Naturally, it can't find the file; it has just written four sectors on top of root directory of the crashed disk (disk address 0-3), which replaced the root directory. So, when Szap goes to look up the file it was copying to (Szap sets the NEW bit in this case), the whole directory has changed and it can't find the copy to file. But, the job is done; the reconstructed old directory is rewritten back to the disk root directory location. Wa la!

Of course, if I had kept a back-up disk, I wouldn't have had to go through all this!

How to UNSAVEP

by Ralph Kenyon

Before I divulge the secret, let me give you a bit of background. BASIC programs can be saved to disk in three ways. One can use the SAVE command, the SAVEF command, or the SAVEP command. SAVE writes the program to disk as a text file. This is great for editing and changing the program, but it is costly in terms of disk space and loading time. BASIC must convert the keywords into token form for its internal use, slowing down the LOAD by a significant factor.

The SAVEF command tells BASIC to write the program to disk in its internal form, that is, in token format. Each keyword only takes up one byte of disk space. But, such a file is not editable or readable.

Poly has always had System and non-System files. The earliest form of protection was to make the file a System file. The earliest BASIC's refused to LIST or SAVE a file that was a system file with the cryptic message "I can't do that to a protected file!" Later, Poly strengthened its protection method by adding the SAVEP command and encrypting each file so saved. But, Poly chose to encrypt the internal token format, and thereby lies their downfall.

How to Undo it.

The first and hardest way is to use brute force. Disassemble BASIC, search for the SAVEP command, and the LOAD command, recover the decription code, and use this to write an asssembly language program to decrypt a program. But, there is an easier way.

When a program is stored in memory, it is stored from MEMTOP down, in reverse order, and in token format. The interpreter reads this program data one byte at a time. A SAVEP, or encrypted program, must be decrypted before the interpreter can read and execute it. When a protected program is running, its unprotected version is in memory. When a program is LOADed, BASIC keeps a flag to remember if it was protected and checks that flag to prevent access to the program code.

Recovering the unencrypted code requires that the flag be set to the value that means "not encrypted". In all versions of Poly BASIC that I have encountered, that means that the flag is set to zero. Of course, one must find where in memory that flag is, and one must find some way to interrupt the program without letting BASIC erase it. Various tricks have been used, and Poly has written protection against those tricks into later versions of BASIC. C04 is the hardest to crack, but not impossible.

Well, space prevents me from finishing the story in this issue, so as much as I hate "TO BE CONTINUED...", here is one for you. There is enough information for enterprising hackers to solve the problem for themselves, but I will go on with the boring details in another issue.

In the Public Domain

This issue we have a new compiler. Abstract Systems has released the Little-Ada compiler system to the public domain.

```
Disk PGL-V-27 has 37 files on it, 24 free entries.
348 sectors in use, 0 sectors deleted, 2 sectors free.
 Size Name.
   12 EXAMPLE.AD     1 ADdef.ED     1 Date.DT     1 INITIAL.CD
    2 ARISE.GO       5 DLIST.GO     5 LICODE.SY   1 Reset.GO
    1 Today.GO       9 REF-CARD.DC  3 HELP.DC     8 CHECKSUM.GO
    4 DUP.GO         5 MIRROR.GO    1 DEMO.CD    95 DOCUMENT.DC
    2 STUFF.CD       9 ANNOUNCE.DC  3 LICODE-INFO.DC
```

Little-Ada implements a minimal subset of Ada that preserves the block structuring and syntax forms in full Ada. ("Ada" is a registered trademark of the U.S. DoD.) Ada has been described as, in part, a block structured language with strong typing that is Pascal based. Little-Ada implements one aggregate (ARRAY), two objects (variable, and constant), three control structures (subprogram, if-then-else, and loop-exit), relational and arithmetic operators and an assignment statement. Arithmetic operators include *, /, MOD, +, and -.

Reserved Words

Little-Ada has 18 reserved words:

```
ARRAY     ELSE   EXIT    LOOP   OF        THEN
BEGIN     ELSIF  INTEGER MOD    PRAGMA    TYPE
CONSTANT  END    IS      NULL   PROCEDURE WHEN
```

There are 18 special symbols in Little-Ada.

```
+   -   >=   :=   *   /
<   >   <=   ..   =   :
(   )   /=   --   ;   _ (underscore)
```

a. Six condition test symbols:

```
=   /=   <   <=   >   >=
```

b. Four arithmetic operators:

```
+   -   *   /   (& MOD)
```

c. Assignment opereator (becomes):

```
:=   (as in NEW := OLD + 1;)
```

d. Object type indicator:

: (as in SIZE : INTEGER;)

e. Range indicator (ellipsis read "to"):

.. (as in ARRAY (1..10) OF)

f. Comment (Terminated by end of line):

--

g. Statement terminator:

;

h. Parenthesis:

( )

i. Underscore:

_ (ignored in numbers)

An ARRAY type may be declared with the following syntax:

```
TYPE NEW_TYPE IS ARRAY (LOW..HI) OF BASE_TYPE;
```

Examples:

```
TYPE VECTOR IS ARRAY (1..10) OF INTEGER;
TYPE MATRIX IS ARRAY (1..10) OF VECTOR;
```

Variables and constants are declared as follows:

```
VARIABLE_NAME : TYPE_NAME;
CONSTANT_NAME : CONSTANT INTEGER := value;
```

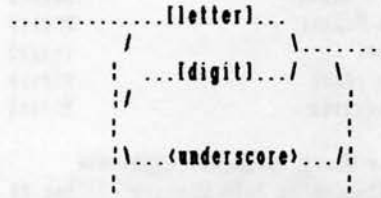
Examples:

```
SIZE : INTEGER;
HEIGHT : INTEGER := 10;
LINES_PER_PAGE : CONSTANT INTEGER := 66;
```

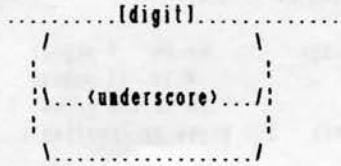
Little-Ada (L/1) Syntax Diagrams

PROGRAM

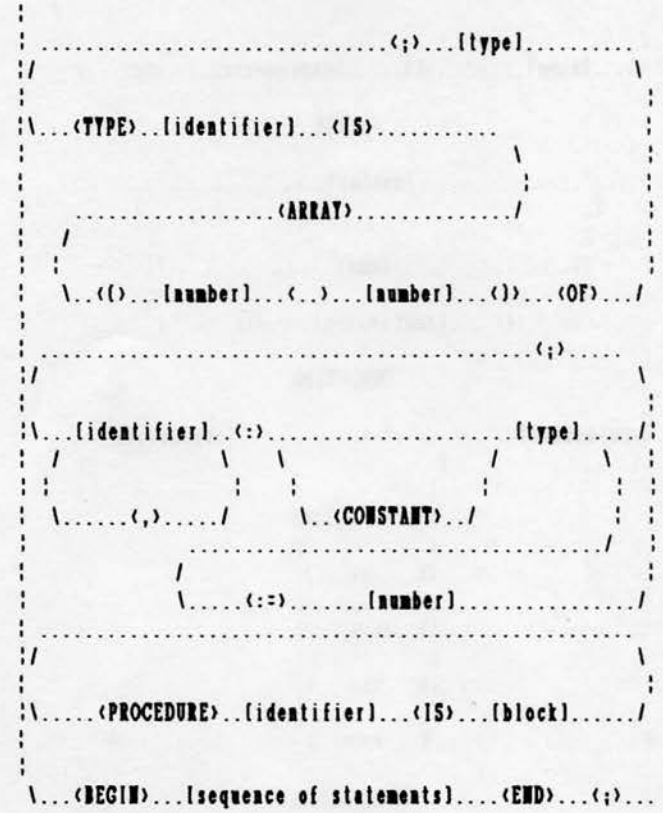
IDENTIFIER



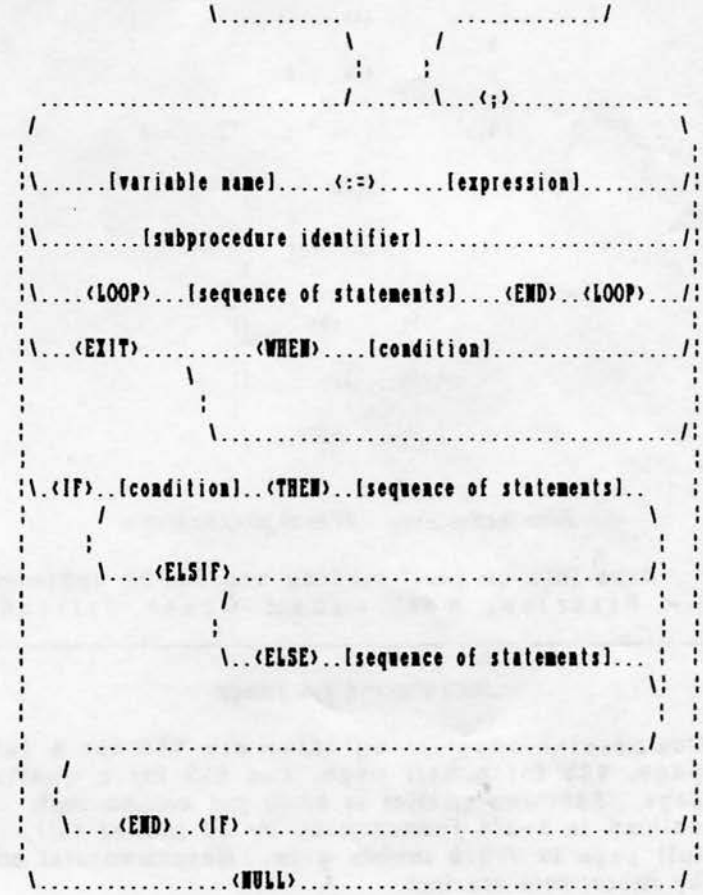
NUMBER

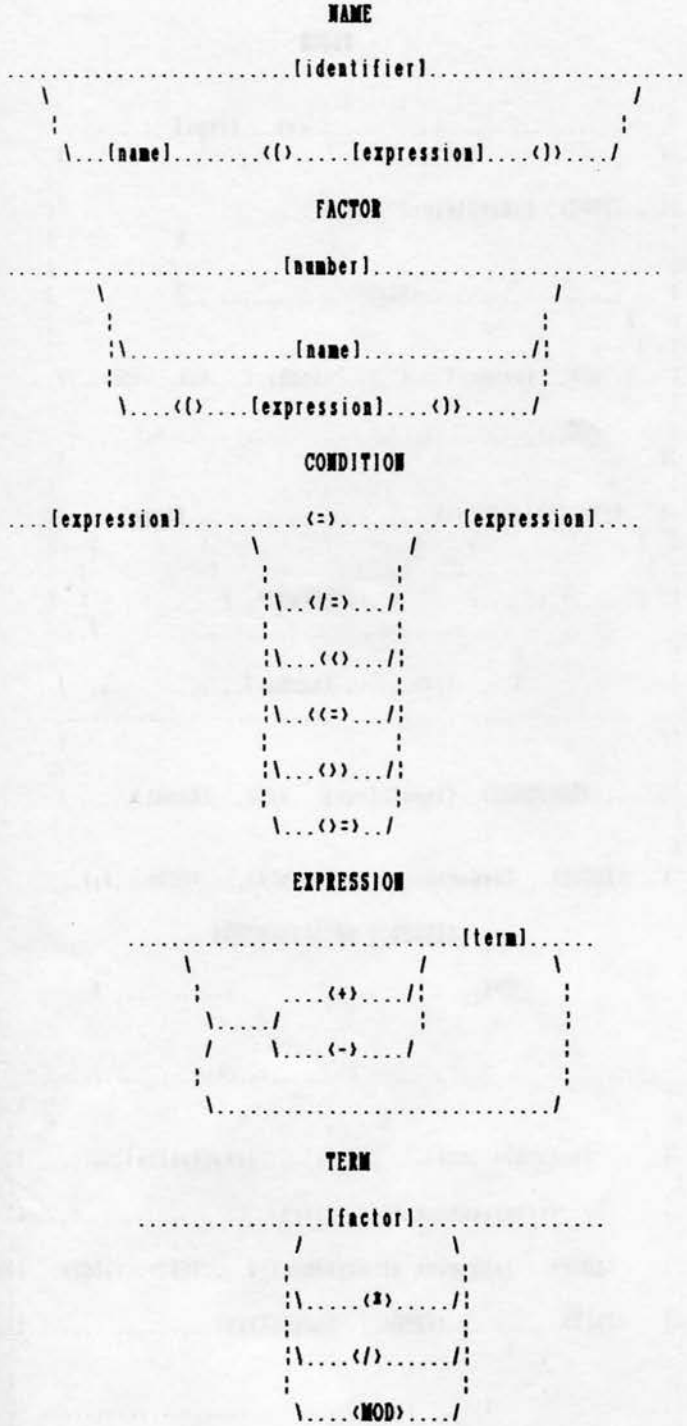


BLOCK



SEQUENCE OF STATEMENTS





Readers Responses

More info on fault finding and fitting upgrades.
-- Briarity, Nottingham Great Britain

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

(Send $1.00 for a complete catalog--(free with any order!.)
(Make check or money order payable to Ralph Kenyon.)

PolyMorphic Systems
7334-H Hollister Avenue,
Santa Barbara, CA 93117

One source for new and reconditioned systems, hard disk sub-systems, parts, conversions, manuals, and service. We've got it all! CALL (805) 685-6238.

For Sale: 8813 Twin system with MS and 10M Priam HD, 8813 DSDD 5" with 10M Priam HD, 3 VTI boards, 1 SD controller, Poly Keyboard & Screen enclosure, 3 Serial cards, 1 Cassette interface, 1 Disk Controller Tester, 20 Boxes 8" disks, Field Service Manual, Dealer's kits, Manuals, etc. $1,500 for the lot. - Charles Trayser, 415-651-5931.

FOR SALE: Two 5" SSSD SA400 drives, $25 each. - Ken Lowe, 5936 W. Zina Circle, West Valley City, UT, 84120 (801) 969-7736.

FOR SALE: Poly 8810 box with power supply and mother board. $50 plus shipping. Charles A. Thompson, 2909 Rosedale Avenue, Dallas, Texas 75205-1532, Phone: (214)-368-8223

The Other BASIC

by Ralph Kenyon

I have begun to convert some Poly BASIC programs into PC-BASIC (actually GW-BASIC, since I don't have a genuine IBM, but a clone). BASICally (heh, heh), I use the try and see technique. I port the Poly program over to the PC through the serial port using ProComm on the PC and the Poly's PRINT command. Once there, I TRY to run the program and SEE if it works. Once I get an error, I go back and change the program on the Poly (I like the Poly Editor better.) I then repeat the process.

Here are some of the things I have learned so far.

1. IBM uses a colon instead of a backslash as a statement separator. Replace every "\" with a ":".

2. IBM BASIC has no MAT statement. Every MAT statement must be replaced with a FOR ... TO ... NEXT loop.

3. IBM BASIC does not require setting the number of bytes in a DIMension statement. IBM BASIC DIM statements are like Poly BASIC versions A00 and A01. DIM A$(20) is enough.

4. FILE access is different. PC BASIC file channels start at 1 rather than 4 as in the Poly. Also, the statement syntax is different. To open a file on channel 1 use: OPEN "I",#1,"NAME". To input from the file use: INPUT #1,I1$. Also, you must test for an end of file condition with the EOF condition. Use: IF EOF(1) THEN (statement number).

Well, I'll continue this later.

BugNotes

Abstract Systems BugNote 013.0 January 18, 1983

ASIN

BASIC C03 has a bug in the ArcSine function. ASIN(-1) returns 1.5707963 vice -1.5707963.

```
$BASIC
System 88 BASIC C03, 04/14/81.  36683 bytes free.
>10 X=0 \D=1/2^10
>20 D=D/2 \REM make delta smaller
>30 A=-1+D \REM make angle -1 plus a small delta
>40 PRINT %15E7,A,ASIN(A) \REM Angle A, and ArcSine A
>50 IF ASIN(A)>0 THEN X=X+1 \REM count for exit
>60 IF X<3 THEN 20 \REM we've seen enough
>RUN
 -9.9999240E-01 -1.5668976E+00
 -9.9999620E-01 -1.5680395E+00
 -9.9999810E-01 -1.5686469E+00
 -9.9999900E-01 -1.5693821E+00
 -9.9999950E-01 -1.5697963E+00
 -9.9999980E-01 -1.5701638E+00
 -9.9999990E-01 -1.5703491E+00
 -9.9999990E-01 -1.5703491E+00
 -1.0000000E+00  1.5707963E+00
 -1.0000000E+00  1.5707963E+00
 -1.0000000E+00  1.5707963E+00
>BYE
(Exec 95 06/12/81)
$Pr NOLOG
```

HELP!

In this section I share with you the help system files I have built up over the last few years. (The entire system is included with Abstract Systems Exec.)

```
HELP COMMAND RENAME

      HELP file for system command "RENAME"

The "RENAME" command renames one existing file.

Syntax:   "RENAME OLD-NAME NEW-NAME" (RETURN)

"RENAME [<n<path<old-file.X1] [<n<path<new-file.X2]"

'n' is the drive number, 'path' is the subdirectory path,
old-file is the old file name, and new-file is the new file
name. When renamed, the new bit is cleared.


"RENAME *.X1 *.X2" renames all files with old extension 'X1' to
       the same name with new extension 'X2'.
"RENAME oldfile.* newfile.*" renames all files with the old
       name 'oldfile' to the new name 'newfile'.

Minimum size: "REN"
Examples: "REN <2<SUB<DATA-1 <2<SUB<DATA-2"
          "REN DATA.* OLD-DATA.*"
          "REN *.DT *.TX"
```

Readers Requests

HELP! I have double sided REMEX 8" drives for my MS, but I don't know how to strap the shunts. Anyone who knows how to configure these drives for the Poly MS please write Karl Thomas, 145 Bond Street, Elk Grove, IL, 60007-1218

Other readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. What happens when a BASIC program is saved with SAVEF, or SAVEP. How to UNSAVEP. How would CP/M be of use. An explanation of the Front Panel. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Hardware update recommendations, Source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

Bit Bucket

Is a two-bit computer a binary device?

According to Doug Schirrippa --- "On a clear disk you can seek forever." Isn't that like the agony of delete?

In This Issue

Coming Soon

Modems and Communications software; More: BASIC for Beginners, how to UNSAVEP protected Programs, System Programmers Notes, Converting Poly BASIC to PC BASIC, Help, BugNotes, Public Domain Software, etc.

Questions

Can you find and answer the questions asked in this issue? Send your answers and requests in.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL

Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter; please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter* are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Overseas $20.00 yr., payable in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).

PolyLetter

Editorial

Well, I was glad to see that my mention of languages prompted Bob Bybee to send me an article on the C language. How about some of our readers sending in more articles to share with everyone. This is your newsletter; let's all participate!

Introduction to C

by Bob Bybee

In PL 8802, Ralph mentioned the C language. I've been using it for some time at work, on Unix-based systems. I'm told it's similar to Pascal, but without the restrictions that Pascal puts on you. (In C it's easier to do your job, or to hang yourself.) I'm not a Pascal person, though, so let's look at a quick example of C versus BASIC.

In BASIC, a portion of a program might look like:

```
100 REM  compute N factorial
110 REM  N must be defined earlier
120 F = 1
130 FOR I = 1 TO N
140 F = F * I
150 NEXT I
160 RETURN
```

In C, the same program could be written:

```
/* compute N factorial */
factorial( n )
    int n;
    {
    int i, f;
    f = 1;
    for (i = 1; i <= n; ++i)
        f = f * i;
    return (f);
    }
```

Let's compare the two. Some differences are immediately noticeable, like the fact that each C statement ends with a semicolon. This allows us to put multiple statements on a line, or extend a statement over several lines. It doesn't matter, as long as we remember the semicolon.

All variables in C must be declared. Most modern languages require this, and it means more typing, but also prevents new variables from being "created" if we misspell something. In this program, n is declared an "int" (integer), as are i and f. Since n is an input to this function, it is declared outside the braces which enclose the function. Variables used within the function are declared within it, and are not visible outside the function. (In BASIC, our use of I might disrupt some other use of I elsewhere in the program.)

This function is named "factorial". In C, every program section is a function. All input/output is done in C using special functions provided with the C compiler. Even the main program is just another function, but it has the special feature that its name must be "main".

C comments begin with the characters '/*' and end with '*/'. They may go on for as many lines as you wish. If you leave off the trailing */, the rest of your program will get "commented out!"

The "for" statement in C looks much like the BASIC FOR .. NEXT loop. It has three parts, separated by semicolons. The parts work as follows Part 1 (i = 1) is executed once, to initialize the loop. Part 2 (i <= n) is a test, to see if the loop should continue. If true, the loop continues to run. Part 3 (++i) is what to do at the end of the loop. Typically this part increments the loop counter. (++i means i = i + 1)

Part of the power of C is that nearly any statement can be placed anywhere. In fact, any series of statements can be placed within braces, and used wherever a single statement would normally go. If we wanted to do a number of things inside the "for" loop, we would write them as:

```
for (i = 1; i <= n; ++i)
    {
    f = f * i;
    printf("f is now equal to %dn", f);
    xvalue = otherfunction();
    }
```

Now we have introduced some more things to explain. First, "printf" is a standard C function which prints. It looks like any other function, in that it is called by giving its name, and its arguments go in parentheses. The first argument to "printf" is always a string in double quotes, which is like a PRINT USING string in BASIC. Its job is to format the other arguments. The '%d' portion of the string specifies one argument to be printed in decimal; you can also do things like %10d to specify a field width. The 'n' portion is shorthand for "newline", C slang for a carriage return and linefeed.

On the next line, we are calling another function named "otherfunction". We will assign its return

value to the variable "xvalue". This illustrates that C allows multi-character variable and function names. It also shows that a function need not have any arguments; "otherfunction" is called without any arguments in its parentheses.

This program listing shows some C traditions, which aren't required but are nearly always used. For one, notice that indentation is used to make the "structure" of the C program clear. Also notice that lower case is used almost exclusively. Variables and function names in C programs can be upper case if you like, but since keywords like "for" must be in lower case, it's easier to make everything lower case.

C has different data types. Besides "int", there is also "short" (small integer), "long" (large integer), "char" (character), and the floating-point types "float" and "double" (double precision). The exact definition of these types varies between different machines. For example, on most IBM-PC C compilers, an "int" and a "short" are both 16-bit integers, but on Unix workstations an "int" is more likely to be 32-bit. "int" is supposed to be the "natural" integer size for any CPU, while "short" and "long" may (or may not) be shorter, or longer, than an int.

C has a very powerful data type called "pointer". A variable can be a "pointer to" almost anything; it does this by containing the memory address of the "thing". You can also define your own data types, called "structures". These allow you to operate on a group of data all at once, keeping it together for simplicity. You can pass an entire structure to a function, if you like. A structure might contain a character string (your name), and two integers (your salary and employee number). Of course, C also has arrays, and they can have any number of dimensions. By using an array of characters, you can do just about anything BASIC can do with a string variable (A$). String manipulations and comparisons can be done with C functions.

You can also define really bizarre things like an array of structures, or a structure containing different arrays, or a structure containing other structures. Believe it or not, such things do come in handy, if you're brave enough to use them. Fortunately, C compilers do a lot of type-checking to make sure that you're doing something that is at least CLOSE to legal.

Our example function returns an integer, f. But functions can return any (simple) data type, or they can return a pointer to anything. We can declare a function which returns a "long" integer by starting it off with

```
long newfunction( arg1, arg2 )
```

This way the compiler knows that "newfunction" returns a "long", instead of the default "int" type.

C also allows you to tell the compiler when to use registers. Programs generally run faster if certain crucial variables are kept in CPU registers as much as possible. C lets you give the compiler a "hint" as to which variables would benefit from this treatment. You do this when you declare the variable:

```
register int i, f;
```

This statement would tell the compiler to use CPU registers for variables i and f, when possible. If the processor only had one register suitable for that data type, only one might get placed in a register. It's left up to the compiler, so different C compilers will do different things. Some newer C compilers will even do this trick without being asked.

Why use C? It is widely available, almost standardized, and creates programs which run fast and consume little memory. There is a great deal of competition among the C compiler people (Microsoft, Borland, and others) to create the fastest C compiler for PCs. And the language is becoming available on most systems, although its roots are in the minicomputer world. The Unix operating system and C are almost synonymous, since they were developed together.

In fact, C was designed as a language in which to write operating systems. This tells us that we can write a lot of machine-dependent code in C. You can do bit manipulations, peek/poke at memory with great flexibility, and do almost anything in C that you can in assembly. The only exceptions I've found are cases where you need to deal with specific CPU registers, or do fancy things with the stack, or handle interrupts. The best way to do these is to write a small "wrapper" routine that does the essential things in assembly, and calls a C routine to do the rest. It's usually easy to make calls between C and assembly, and the interface should be documented with any C compiler. Many C compilers come with an assembler too, incidentally, but not always a great one.

No compiled language will produce as small or as fast a program as you could by hand, in assembly, but I've been impressed at the job a good C compiler will do. Since C is a "high-level" and "structured" language, it's a great, great, great deal easier to write error-free code in C than in assembly. And I've successfully taken a number of programs from my Unix system at work, downloaded the C source code to my IBM-PC at home, and recompiled it there. That's a useful trick you absolutely can't do in assembly language.

A number of introductory C books are available in most bookstores. C compilers for PC-compatibles start at under $50. Small-C is available for the Poly and for many other small computer systems, but may lack some features (floating-point data types, structures, etc.). Try C - I think you'll like it.

[Small C for the Poly is available from PolyLetter on the public domain disk PolyGlot Library Volume eleven (PGL-V-11) for $6.00 -- Ed.]

Front Panel Bytes

by Ralph Kenyon

One reader asked for an explanation of "What exactly do the two-digit numbers in memory represent". Other readers have asked for an explanation of the front panel.

A typical Front Panel Display looks like this:

```
PC 05F1 E1 FB 76 E5 3A 88 2D B7
SP 0FEA 3F F1 05 5B FC 55 F6 2E
HL 634A 42 37 0D 0D 0D 7B 70 61
DE 0020 14 0C E9 F5 C5 D5 E5 2A
BC 0040 1C 0C E9 D3 08 21 00 0C
AF 3F56   Z       ↑

A4F5    20 6F 6E 65 20 6D 6F 72
A4FD    65 20 74 68 61 6E 0D 79
A505    6F 75 20 6E 65 65 64 2C
A50D    20 61 6E 64 20 75 73 65
A515  → 20 70 72 69 6E 74 20 66
A51D    6F 72 6D 61 74 20 73 74
A525    61 74 65 6D 65 6E 74 73
A52D    20 74 6F 20 73 65 74 20
```

Except for one special place where the letters "C", "M", or "Z" may appear, each character can only be one of the digits (0-9) or one of the first 6 letters of the alphabet (A-F). These sixteen characters are the sixteen hexadecimal digits.

Most of us can only count by tens, or what is known as 'decimal arithmetic'. In decimal arithmetic there are ten digits (0-9), and a number is expressed using a positional notation. For example, a three digit number has one digit in the hundreds place, one digit in the tens place, and one digit in the units place. The key to using this knowledge for understanding hexadecimal arithmetic is to notice that the digit in the tens place tells us how many tens are to be counted. One ten is 10^1, and one hundred is 10^2; also one unit is 10^0. The three digit decimal number XYZ can be expressed as

X times 10^2 + Y times 10^1 + Z times 10^0

where "10" represents the base, which is ten, and X, Y, and Z are digits.

In hexadecimal arithmetic there are not ten, but sixteen digits and "10" would represent the base, which is sixteen. To avoid confusion, a number which is not in decimal notation is usually written with a decimal notation subscript, and is pronounced differently than decimal numbers. "10 base 16" is not pronounced "ten"; it is pronounced "one-zero" or "one-zero-base-sixteen". If no confusion results, the subscript is often dropped.

Why use hexadecimal around computers? Hexadecimal is a good short hand for binary numbers, which can be very long. "What's binary?", you say? Well, the binary number system is the system one gets when there are only two bases, (0 and 1). The reason binary is useful around computers is that the computer is a large collection of glorified switches which can be either on or off. The binary number system is ideal for representing these on or off conditions because it has only two digits and we can assign the 1 to on and the 0 to off (or vice versa, as long as we stick to one system).

A word about counting. We count by going through the digits one at a time until we come to the last one. Then we say "one zero". If we were being precise, we would have started our count with the number "zero-one", or "zero-zero-one" (which are the same number, namely "1", but different names). In decimal counting we would have counted zero-zero-one, zero-zero-two, zero-zero-three, zero-zero-four, zero-zero-five, zero-zero-six, zero-zero-seven, zero-zero-eight, zero-zero-nine, one-zero, one-one, and so forth, until we got up to zero-nine-nine. Then we would go to one-zero-zero and continue.

Counting in binary is easier because there are only two digits. So, the count goes zero-zero-one, zero-one-zero, zero-one-one, one-zero-zero, one-zero-one, one-one-zero, one-one-one, and so forth. Counting in hexadecimal is harder because there are sixteen bases to remember, and when the digits 0-9 are used up one then starts on the digits A-F.

Here is a table comparing the first sixteen numbers

name /Notation	decimal	hexadecimal	binary	
zero	00	00	00000	
one	01	01	00001	= 1x2^0
two	02	02	00010	= 1x2^1
three	03	03	00011	
four	04	04	00100	= 1x2^2
five	05	05	00101	
six	06	06	00110	
seven	07	07	00111	
eight	08	08	01000	= 1x2^3
nine	09	09	01001	
ten	10	0A	01010	
eleven	11	0B	01011	
twelve	12	0C	01100	
thirteen	13	0D	01101	
fourteen	14	0E	01110	
fifteen	15	0F	01111	
(sixteen)	16	10	10000	= 1x2^4

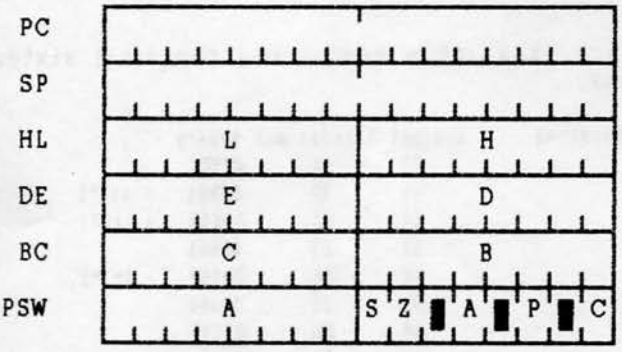
These counting methods are used to represent numbers in computers. The Poly has an 8080 as its Central Processor Unit (CPU); the 8080 is an eight bit machine. (A BIT is a BInary digiT.) The 8080 CPU processes eight bits at a time. Those eight bits can be represented by an eight digit binary number. The binary number 00000000 would represent all eight bits being off, and 11111111 would represent all eight bits being on.

Notice in the table that each of the four bit binary numbers corresponds to one of the hexadecimal digits. By using hexadecimal notation we can represent each eight bit quantity with two hexadecimal digits. 00000000 binary is the same as 00 hexadecimal, and 11111111 binary is the name as FF hexadecimal. An eight bit number is called a "byte" and, in the context of the 8080, two bytes is called a word. Now, the Poly's memory is eight bits

wide, naturally, since the 8080 is an eight bit machine. So, it is easy to represent the contents of memory using hexadecimal bytes.

To say what those contents represents requires an understanding of the architecture of the 8080 CPU. The 8080 is a little like a desk with several baskets on it. Each basket is called a register. Let's draw a picture of the 8080 and all its registers. Some registers are sixteen bits, or a full word wide, and some are only eight bits (one byte) wide. There are three sixteen bit general purpose registers which can also be organized as six eight bit registers. The sixteen bit registers are the program counter (PC) and the stack pointer (SP); the three general purpose registers are registers BC, DE, and HL, which can also be accessed as 6 eight bit registers B, C, D, E, H, and L. There is also an accumulator (A) which is strictly an eight bit register. Finally, there are the five one bit flags. These flags, together with the eight bit accumulator make up the program status word (PSW). The layout of this drawing corresponds to the layout used by the front panel display.

8080A Architecture



The program counter (PC) is a sixteen bit register which is understood as containing the address in memory of the next machine language instruction to be executed. The 8080 uses the contents of the PC to fetch the next instruction from memory. It then executes that instruction. Register HL, taken as a sixteen bit register, is called the data pointer and is understood as pointing to a memory location where data is stored. Some of the 8080 instructions operate on this data. When this data is referred to, the symbol "M" is used, and can be thought of as the Memory location HL points at.

The largest sixteen bit binary number, 1111111111111111, can be represented in hexadecimal as FFFF and its decimal value is 65,535. 2^10 is 1024 and is often referred to as a K. Since 2^10 divides into 65,536 exactly 64 times, that is where the 64K comes from. Because of the sixteen bit size of the registers, the 8080 can only accesss 64K of memory or address space.

Notice that what the PC points at is considered to be an instruction for the 8080, while what register HL points at is considered to be data. What both point at is a location in memory whose contents are just an eight bit binary value.

How do we tell the difference between instruction and data? Good question. We can't tell by the contents or value alone. An eight bit binary number could be an 8080 instruction, or it could be data. It depends upon where the PC and HL are pointing.

Before saying what values are what instructions, let me take a moment to say what the 8080 can do. The things that the 8080 can do are perform arithmetic and logical operations on the accumulator (A), move data from register to register or to or from a register and memory, add a sixteen bit register to HL, or change the program counter by CALLing a subroutine (like a BASIC GOSUB statement), or JuMPing to a new location (like a BASIC GOTO statement). There are also conditional CALL's and JMP's, depending upon the value of the flags.

One of the most common instructions is the move instruction (MOV). An eight bit value of the form 01DDDSSS is an instruction to move from an eight bit register to an eight bit register where DDD is the destination and SSS is the source. The three-bit values for the source and destination are: 000 = B, 001 = C, 010 = D, 011 = E, 100 = H, 101 = L, 110 = M, and 111 = A, where "M" refers to the location pointed at by HL. The one exception is 01110110, which is the halt (HLT) instruction.

01111000 is the binary representation of the instruction which moves the contents of register B into the accumulator. The front panel displays in hexadecimal, and this value would be 78 hexadecimal. Remembering a string of 0's and 1's and what each one stands for is almost impossible. Even remembering them as hexadecimal numbers is nearly as difficult. There are 256 (decimal) different possibilities, and almost all of them are valid instructions.

To simplify the memorization problem, mnemonic names are used to represent, in symbols, the instructions, and an assembler is used to convert the symbols to machine readable form. The symbolic representation is called assembly language. The assembly language representation of the instruction which moves the contents of register B to the accumulator is "MOV A,B". All 63 different move instructions can be represented in the form "MOV D,S". Table 1 is a conversion chart for converting hexadecimal to assembly language. Table 2 is a list of assembly language instructions with their explanations.

Remember that register pair HL is a data pointer? In this case the contents of memory might not be instructions, but could be data in many forms. One form is ASCII text ASCII stands for American Standard Code for Information Interchange. The ASCII standard codes 128 characters and control codes. For example, the letter "A" is represented by the binary number 1000001, which is a hexadecimal 41, and a decimal 65. Table 3 is a listing of all 128 ASCII characters. Of course, the data could be anything else as well; it depends upon the program which is using the data.

Table 1: Converting hexadecimal to 8080 assembly language instructions

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00	NOP	LXI B,w	STAX B	INX B	INR B	DCR B	MVI B,b	RLC	---	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,b	RRC	00
10	---	LXI D,w	STAX D	INX D	INR D	DCR D	MVI D,b	RAL	---	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,b	RAR	10
20	---	LXI H,w	SHLD	INX H	INR H	DCR H	MVI H,b	DAA	---	DAD H	LHLD	DCX H	INR L	DCR L	MVI L,b	CMA	20
30	---	LXI SP,w	STA	INX SP	INR M	DCR M	MVI M,b	STC	---	DAD SP	LDA	DCX SP	INR A	DCR A	MVI A,b	CMC	30
40	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A	40
50	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A	50
60	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A	60
70	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A	70
80	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	80
90	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	90
A0	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	A0
B0	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	B0
C0	RNZ	POP B	JNZ	JMP	CNZ	PUSH B	ADI	RST 0	RZ	RET	JZ	---	CZ	CALL	ACI	RST 1	C0
D0	RNC	POP D	JNC	OUT	CNC	PUSH D	SUI	RST 2	RC	---	JC	IN	CC	---	SBI	RST 3	D0
E0	RPO	POP H	JPO	XTHL	CPO	PUSH H	ANI	RST 4	RPE	PCHL	JPE	XCHG	CPE	---	XRI	RST 5	E0
F0	RP	POP PSW	JP	DI	CP	PUSH PSW	ORI	RST 6	RM	SPHL	JM	EI	CM	---	CPI	RST 7	F0
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	

Rotate Instructions

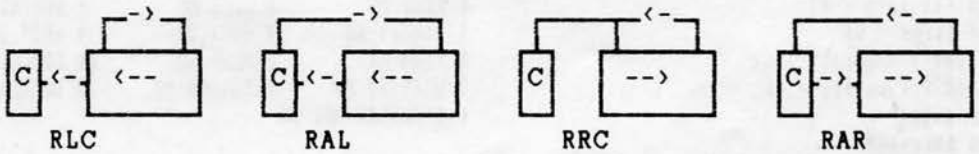


Table 2:

CODE	Interpertation
ACI b	add next byte with carry to accumulator
ADC R	add register R into acc with carry
ADD R	add register R into accumulator
ADI b	add next byte into accumulator
ANA R	AND register R into accumulator
ANI b	AND next byte into accumulator
CALL w	call unconditional
CC w	call on carry (if carry = 1)
CM w	call on minus (if sign = 1)
CMA	complement accumulator
CMC	complement carry flag
CMP R	compare reg R to accumulator
CNC w	call on no carry (if carry=0)
CNZ w	call on not zero (if zero = 0)
CP w	call on positive (if sign = 0)
CPE w	call on parity even (if P = 1)
CPI b	compare next byte to accumulator
CPO w	call on parity odd (if P = 0)
CZ w	call on zero (if zero = 1)
DAA	decimal adjust accumulator
DAD RP	add register pair to register HL
DCR R	decrement register R by 1
DCX RP	decrement register pair RP by 1
DI	disable interrupts
EI	enable interrupts
HLT	halt processor. (Stop dead!)
INX RP	increment register pair RP by 1
IN b	input from port specified by next byte
INR R	increment register R by 1
JC w	jump on carry (if carry = 1)
JM w	jump on minus (if sign = 1)
JMP w	jump unconditional
JNC w	jump on no carry (if carry=0)
JNZ w	jump on not zero (if zero = 0)
JP w	jump on positive (if sign = 0)
JPE w	jump on parity even (if P = 1)
JPO w	jump on parity odd (if P = 0)
JZ w	jump on zero (if zero = 1)
LDA w	load acc from memory adr by next word
LDAX RP	load acc from memory adr by reg pair
LHLD w	load registers H and L direct
LXI RP,w	load register pair with next two bytes
MOV D,S	move register S to register D
MVI R,b	move next byte byte to register
NOP	no operation
ORA R	OR register R into accumulator
ORI b	OR next byte byte with accumulator
OUT n	output to port specified by next byte
PCHL	move reg HL to program counter
POP RP	pop registers pair off stack
PUSH RP	push registers pair on stack
RAL	rotate accumulator left thru carry
RAR	rotate accumulator right thru carry
RC	return on carry (if carry =1)
RET	return
RLC	rotate accumulator left
RM	return on minus (if sign = 1)
RNC	return on no carry (carry = 0)
RNZ	return on not zero (if zero = 0)
RP	return on positive (sign = 0)
RPE	return on parity even (if parity = 1)
RPO	return on parity odd (if parity = 0)
RRC	rotate accumulator right
RST N	restart at address 00nnn000
RZ	return on zero (if zero = 1)
SBB R	subtract reg R from acc with borrow
SBI b	subtract next byte from acc with borrow
SHLD w	store register HL direct
SPHL	move register HL to stack pointer
STA w	store accumulator in address
STAX RP	store acc in memory adr by reg pair
STC	set carry flag to 1
SUB R	subtract register R from accumulator
SUI b	subtract next byte from accumulator
XRA R	XOR register R into accumulator
XTHL	exchange top of stack with register HL
XCHG	exchange register HL with register DE
XRI b	XOR next byte into accumulator

Table 3: ASCII - HEX - Decimal

CODE	HEX	DEC	CODE	HEX	DEC	CODE	HEX	DEC	CODE	HEX	DEC
NUL	00	0	SPACE	20	32	@	40	64	'	60	96
SOH	01	1	!	21	33	A	41	65	a	61	97
STX	02	2	"	22	34	B	42	66	b	62	98
ETX	03	3	#	23	35	C	43	67	c	63	99
EOT	04	4	$	24	36	D	44	68	d	64	100
ENQ	05	5	%	25	37	E	45	69	e	65	101
ACK	06	6	&	26	38	F	46	70	f	66	102
BEL	07	7	'	27	39	G	47	71	g	67	103
BS	08	8	(	28	40	H	48	72	h	68	104
HT	09	9	)	29	41	I	49	73	i	69	105
LF	0A	10	*	2A	42	J	4A	74	j	6A	106
VT	0B	11	+	2B	43	K	4B	75	k	6B	107
FF	0C	12	,	2C	44	L	4C	76	l	6C	108
CR	0D	13	-	2D	45	M	4D	77	m	6D	109
SO	0E	14	.	2E	46	N	4E	78	n	6E	110
SI	0F	15	/	2F	47	O	4F	79	o	6F	111
DLE	10	16	0	30	48	P	50	80	p	70	112
DC1	11	17	1	31	49	Q	51	81	q	71	113
DC2	12	18	2	32	50	R	52	82	r	72	114
DC3	13	19	3	33	51	S	53	83	s	73	115
DC4	14	20	4	34	52	T	54	84	t	74	116
NAK	15	21	5	35	53	U	55	85	u	75	117
SYN	16	22	6	36	54	V	56	86	v	76	118
ETB	17	23	7	37	55	W	57	87	w	77	119
CAN	18	24	8	38	56	X	58	88	x	78	120
EM	19	25	9	39	57	Y	59	89	y	79	121
SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
FS	1C	28	<	3C	60	\	5C	92	:	7C	124
GS	1D	29	=	3D	61	]	5D	93	}	7D	125
RS	1E	30	>	3E	62	^	5E	94	~	7E	126
US	1F	31	?	3F	63	_	5F	95	DEL	7F	127

In the Public Domain

Disk PGL-V-28 has 32 files on it, 24 free entries.
350 sectors in use, 0 sectors deleted, 0 sectors free.

Size	Name.	Size	Name.	Size	Name.
43	GRAPHICS.BS	1	GRAPHICS.DC	2	BIGPRINT.DC
12	BIGPRINT.BS	8	BIGPRINT.DT	2	DEMO.DC
6	DRAWLINE.BS	3	FLAG.BS	4	CENSOR.BS
6	XMAS-TREE.BS	23	HEADER.BS	9	CASTLE.BS
28	ANSWERS.DT	19	SCROLLING.BS	14	NUMBERS.BS
23	STATES.BS	5	THE-FLY.BS	24	STATES_&_CAPITALS.BS
4	HILO.BS	1	DATE.DT	1	DECIMAL-MATH.DC
4	FLAG/1.BS	17	QUIZ.BS	4	HEAT_MULTIPLIER.BS
6	DEMO.DT	6	DEMO.BS	24	SPELLING-BEE.BS
7	MYSTERY.BS	5	LESSON.BS	32	DECIMAL-MATH.BS
1	SPELLING-BEE.DC				

This disk contains a collection of novelty and

educational programs GRAPHICS is 6 programs that create pictures: Diamond, Old Glory, Snoopy, Poly-88, Playboy Bunny and a" Christmas tree. BIGPRINT prints big letters; it shouldn't be too difficult to lift the writing portion of the program for use in any application that requires large, easy to read characters. The program DEMO has data derived directly from the Character Generator ROM on the video board and thus has an exact copy of the font the system uses. CENSOR deletes those #!?%# BAD words. HEADER creats an eight character mast-head. CASTLE is a date keeping program with a unique visual approach. SCROLLING demonstrates sideways scrolling in graphics. NUMBERS and DECIMAL-MATH are math drill programs. QUIZ proports to measure your creativity. STATES_&_CAPITALS gives you a multiple-choice quiz, while STATES will print the name of a state and ask you for the capital or give you the name of a state capital and ask you for the name of the state. THE-FLY has to be guided out of a maze with your help. HILO guesses your number.

Readers Responses

More info on fault finding and fitting upgrades.
-- Briarity, Nottingham Great Britain

How to UNSAVEP

by Ralph Kenyon

CONTINUED from PolyLetter 8802...

You recall I told you that a BASIC program is stored in memory from the top down in reverse order, and that BASIC keeps a flag to remember whether or not the program was protected (saved in SAVEP mode). Well, the simplest way to recover such a program is to push the LOAD button when the program is running. The program remains in memory and can be recovered by GETting BASIC and REEntering. The new copy of BASIC will discover that there is a program in memory, but will not know that it was protected.

This procedure works only if no user programs have been run by an INITIAL file. When I am decrypting SAVEP programs, I rename my INITIAL file to initial and re-boot the system first. Then, when I press the LOAD button, nothing happens to change what was in memory. This procedure does not work for all versions of BASIC. It depends where the protect flag is in memory. In the versions where it does work the protect flag had to have been in the area of the BASIC.GO file which is used for data. If the protect flag is further up in memory than BASIC.GO uses, then the new copy of BASIC will still know about the protected status. Fortunately, there is a way to find that flag, but as you readers of mysteries know this is the perfect time for another one of those cliff-hangers. TO BE CONTINUED...

The Other BASIC

by Ralph Kenyon

Poly BASIC uses INP(1) to get one character from the keyboard buffer. INP(1) waits until a character is typed, while INP(0) returns 0 if no character is waiting and 1 if a character is waiting. IBM and GW BASIC uses the variable INKEY$ in place of both of these. Here is how INKEY$ works in terms of Poly BASIC.

```
DEF FN INKEY$(Z)
IF INP(0)<>0 THEN RETURN CHR$(INP(1)) ELSE RETURN ""
FN END
```

To use INKEY$ in place of INP(1) one would need the code

```
100 A$=INKEY$ : IF LEN(A$)=0 THEN 100
```

```
There is no way to directly substitute for INP(0), since
INKEY$ returns a character if it is waiting.
I have programs which use INP(0) and then PEEK in the
keyboard buffer to see if the character typed is an ESCAPE.
If it was not an escape, then the program goes to an INPUT
statement, otherwise, it throws away the character.
That sequence would be coded:
```

```
100 A$=INKEY$ : IF LEN(A$)=0 THEN 100
110 IF ASC(A$)=27 THEN GOTO 200
120 INPUT B$ : C$=A$+B$
130 REM INPUT routine here
...
200 REM ESCAPE routine here
```

The one thing to remember using INKEY$ is that the character has been taken from the keyboard buffer, so you must save it in a variable until you need it.

It isn't safe just to test if ASC(A$)=0 because INKEY$ returns a two character string, with the first character being ASCII 0 in the case a function key or extended character code is typed.

Well, I'll continue this later.

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

PolyMorphic Systems
7334-H Hollister Avenue,
Santa Barbara, CA 93117

One source for new and reconditioned systems, hard disk sub-systems, parts, conversions, manuals, and service. We've got it all! CALL (805) 685-6238.

FOR SALE: Poly 8810 box with power supply and mother board. $50 plus shipping. Charles A. Thompson, 2909 Rosedale Avenue, Dallas, Texas 75205-1532, Phone: (214)-368-8223

For Sale: 8813 Twin system with MS and 10M Priam HD, 8813 DSDD 5" with 10M Priam HD, 3 VTI boards, 1 SD controller, Poly Keyboard & Screen enclosure, 3 Serial cards, 1 Cassette interface, 1 Disk Controller Tester, 20 Boxes 8" disks, Field Service Manual, Dealer's kits, Manuals, etc. $1,500 for the

lot. - Charles Trayser, 415-651-5931.

Abstract Systems, etc.
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

DISKS - MODEMS - PROMS - SOFTWARE - SPELL

1. MAXELL diskettes: 5-1/4" 10 hard sector -- $15 per box.
2. HayesSys modem software (for the Micromodem 100) $25.
3. (A;S) Spell, a good spelling checker for $35.
4. Abstract Systems Exec (Enhancements & bugs corrected) $35.
5. Abstract Systems Proms (Enhancements & bugs corrected) $35.
6. PolyGlot Library Volumes 1 thru 26, $6 each.

- USED Manuals - USED Manuals - USED Manuals - USED Manuals -
These are used manuals received with second hand systems. Most are in good condition, but there may be notes written in them. All are without binders except where noted.
(First come first served)

Q	Name or description	part #	pgs	Price
3	Users Manual (Release 1 Exec/4D)	(none)	183	5.00 old
6	Users Manual (Release 2 Exec/73)	810140	167	5.00 old
5	BASIC A Manual (Release 1)	(none)	116	3.50 old
5	BASIC B Manual (Release 2)	810140	159	4.50 old
1	BASIC C Manual (Release 3)	810162	196	9.50 current
1	BASIC C Reference Guide	(none)	46	2.50 current
4	WordMaster I Manual	(none)	124	3.50 old
1	WordMaster II Manual	810179	181	10.00 current
1	Plan Version 1.0 Manual	(1978)	67	2.00 old
3	Macro 88 Assembler Manual	(none)	60	5.00 current
2	System Programmers Guide (Old)	810133	112	3.00 old
3	BPRINT manual	(none)	16	.50 old
3	System 88 Confidence Manual	810167	43	2.00 current
3	8810/13/DS Confidence Manual	810150	43	2.00 current
1	Field Service Manual	810148	144	10.00 current
3	Poly-88 Assembly and Testing	(none)	108	7.50 old
3	Poly-88 Operation	(none)	128	inc
3	Poly-88 Manual Binder	(none)	-	inc
8	Printer Interface Manual	(none)	23	1.00 current
4	Printer/40 errata sheet	(none)	2	.50 current
4	Cassette Interface Manual	(none)	46	2.00 current
2	Video Terminal Interface Manual	(none)	75	2.50
2	Video Operation	810115	42	1.50
2	Keyboard II Manual	(none)	6	.50
1	Keyboard III Manual	809013	6	.50
1	88 MS Users Manual	810157	21	1.00 current
2	8K RAM Manual	(none)	17	.50
1	16K RAM Operation	810119	2	.50
1	48K RAM Operation	810182	4	.50
	xxx by Chuck Thompson (USED) xxx			
2	PolyGrip Addendum to Poly Manuals	Dec 79	16	.50 old
1	Addendum to Poly Manuals	May 81	42	1.50 current
1	Format Command Summary	May 81	11	.50
1	Don Moe's Bag of Tricks	May 80	7	.50

Manual Shipping: Up to 7 pages: .50
8 to 15 pages: .75
16 to 100 pages: 1.00
each additional 100 pages or fraction + .25

(Send $1.00 for a complete catalog--[free with any order].)
(Make check or money order payable to Ralph Kenyon.)

BugNotes

Abstract Systems BugNote 015.0 March 29, 1983

SIN (and COS)

BASIC C03 has a bug in the SIN (and COS) function. SIN(PI/2) returns .99999986 vice 1. Since the relation between COS and SIN is: COS(x) = SQRT(1-SIN(x)^2), Testing COS seemed appropriate. The same bug also makes COS(0) return .99999986 vice 1. A little experimenting revealed that the error depended upon the number of DIGITS specified. At DIGITS 20 .99999999999999999981 is the result returned. Also, by setting DIGITS to 7 the correct result of "1" was obtained.

Do not expect to get the full number of DIGITS of accuracy when using SIN and COS functions. To minimize the effect of this error, use 5 more DIGITS for calculation than needed for the results. (Where do I get the 5? Notice that the last two digits were wrong at both 8 and 20 specified digits. That's 2. Also, my Texas Instruments calculator use 13 digits of internal calculation for a 10 digit display. That's 3 more.)

```
$BASIC
System 88 BASIC C03, 04/14/81.  38219 bytes free.
>DIGITS 20
>PRINT SIN(PI/2)
 .99999999999999999981                    (<- wrong)
>PRINT COS(0)
 .99999999999999999981                    (<- wrong)
>PRINT SIN(0)
 0                                        (<- correct)
>PRINT COS(PI/2)
 0                                        (<- correct)
>DIGITS 8
>PRINT SIN(PI/2)
 .99999986                                (<- wrong)
>PRINT COS(0)
 .99999986                                (<- wrong)
>DIGITS 7
>PRINT SIN(PI/2)
 1                                        (<- correct)
>PRINT COS(0)
 1                                        (<- correct)
```

Abstract Systems BugNote 016.0 April 5, 1983

Edit 3.3 (6/10/81)

Edit 3.3 has a bug in the read routine. When the number of characters in the file is an exact multiple of 256, Edit thinks there is more to read in the input file. Edit responds with a warning, that there is more text in the input file, and CTRL-A should be used to read more text in. This bug causes no serious problems, but may cause one to wonder if there's something wrong when editing a small file. It just happened to me with a file of only 2 sectors. Nothing serious, just a minor annoyance.

HELP!

In this section I share with you the help system files I have built up over the last few years. (The entire system is included with Abstract Systems Exec.)

```
$HELP COMMAND DELETE

      HELP file for system command "DELETE"

The "DELETE" command deletes a file from a directory.

Syntax: "DELETE [<n<path<file.TX]" (RETURN)  (see UNDELETE)
'n' is a drive number and file is a text file.

"DELETE [file]" deletes a file on the system resident drive.
"DELETE [<n<file]" deletes a 'file' on drive 'n'.
"DELETE [<n<path<file]" deletes a 'file' on drive 'n' and
      in subdirectory 'path'.

Minimum size: "DE"      Example "DE <2<LETTERS<Dick"

$HELP COMMAND PACK

      HELP file for system command "PACK"

The "PACK" command recovers the space all deleted files on a
disk, and removes the deleted file names from the directory.
"PACK" may be combined with "ZAP", yielding "ZPACK".

"PACK" packs the system resident drive.
"PACK n" packs drive 'n'.

Syntax: "PACK [n]" (RETURN)  (see ARISE, and ZAP)

Minimum size: "PAC" or "ZP"     Example "PAC 2"

$HELP COMMAND ZAP

      HELP file for system command "ZAP"

The "ZAP" command zeros all user memory thru MEMTOP.

"ZAP" may be combined with the other system commands "COPY",
"GET", and "PACK". The combined commands are "ZCOPY", "ZGET",
and "ZPACK". Each of these commands first ZAPs memory and
then performs the usual function.

Syntax: "ZAP" (RETURN)          (see GET, COPY and PACK)

Minimum size: "ZA"      Example "ZA"

$HELP PROGRAM CLEAN

      HELP file for system program "CLEAN"

"CLEAN.GO" reinitializes any directory of a disk.

Syntax: "CLEAN n[<path]" (RETURN)

"CLEAN n" cleans the root directory on drive "n".
"CLEAN <n<dir" cleans directory "path" on drive "n".

Example "CLEAN 2"

            (The system must be in ENABLE mode.)
            (The system resident drive cannot be "cleaned".)

$HELP PROGRAM Wait
      HELP file for system program Wait

Wait.GO is a program for use in a command file which waits for
a key-stroke before going on to the next command.
;
Wait Please insert the backup disk in drive 3
;
The (A;S) version works without changing user memory and can be
used in command files which exit from a user program, and then
resume the user program without losing the runtime environment.

For example: (In BASIC)
Please Change the disk in drive 2...
STOP in line 1000
>EXEC
$Wait   (while a disk is being changed)
$CON
>CONTINUE
Thank you.
```

System Progarmmers Guide

```
Symbol name:    Msg

Single value:   040C
Twin value:     E006

Entry:
      HL:       Address of text string delimited by 00 byte

Exit:
      HL:       Points to 00 byte
      A:        00
```

Description:

Msg displays the message pointed to by HL on the screen by calling WH1 with each character and incrementing HL until a 00 byte is encountered. Here is the coding for Msg:

```
Msg     MOV     A,M     ; get chr
        ORA     A       ; done yet?
        RZ              ; return if so
        CALL    WH1     ; display
        INX     H
        JMP     Msg     ; do another one.
```

Readers Requests

Readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. How would CP/M be of use. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Hardware update recommendations, Source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

Bit Bucket

Is a two-bit computer a binary device? Yes and no. 'Binary device' is the term used to describe two military weapon systems.

The first is a chemical warfare weapon whose chemical is too deadly to risk handling. The weapon has two chemicals which are mixed only when the weapon is used. When the two components are mixed the final, deadly, chemical is produced at the site of impact. We (the U.S.) are now in the process of disposing of a large stockpile of such devices.

The second is an obsolete nuclear fission weapon which has two separate masses of fissionable material. Fissionable material is such that when you get a large enough amount of it together, it automatically goes boom. How big this amount is depends upon which kind of fissionable material (Uranium or Plutonium) and is called the 'critical mass'. (It also depends upon the degree to which the immediate environment reflects or supplies neutrons.) In a binary device, each mass is smaller than the critical mass, but taken together are bigger than the critical mass. Jamming the two parts together very quickly produces a prompt critical reaction and everything for miles around goes BOOM.

A two-bit computer is a binary device where 'binary' is an adjective modifying the improper noun 'device' but a two-bit computer is not a binary device, where 'binary device' is a noun.

In This Issue

Coming Soon

Managing Memory by Bob Bybee; Modems and Communications software; More: BASIC for Beginners, how to UNSAVEP protected Programs, System Programmers Notes, Converting Poly BASIC to PC BASIC, Help, BugNotes, Public Domain Software, etc.

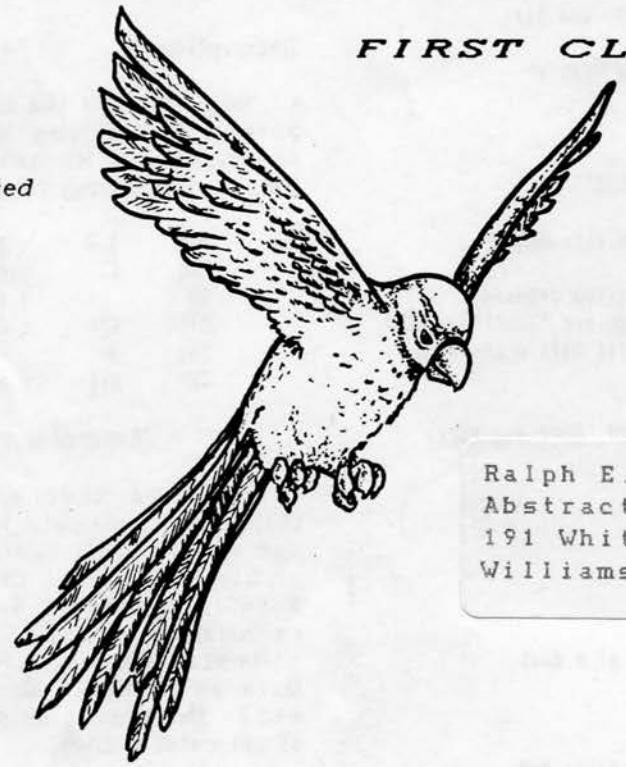
Questions

Can you find and answer the questions asked in this issue? Send your answers and requests in.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL



Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter; please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter* are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Overseas $20.00 yr., payable in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).

PolyLetter

Editorial

When the Poly was first introduced memory was expensive and users often were constrained by budgets to live within 16K of memory. 24K was a dream. Writing larger and more complex programs brought one up against the trade-off between buying more expensive memory and squeezing a few more bytes out of the code. And, what about the poor users who just could not buy more memory? Well, If the program might be marketed, then one would choose to optimize it and squeeze out a few more bytes to make it fit in 16K (or 32K in later years). This is one of the reasons so many of the old BASIC programs have no documentation, no spaces, and long multiple statement lines. Every space eliminated saves 1 byte and every two lines converted to a multiple statement line saves one byte by eliminated the 2 bytes of the line number while adding only one byte for the statement separator. CHAINing in BASIC was also a way to conserve memory.

One of the advantages of learning to program on the Poly is that we know we are limited in memory and disk space, and must be 'byte conscious' when we write programs. Sometimes 'dirty' programming tricks are used to save a few bytes, but mostly we learn to effectively use our limited resources by optimizing our programs so that they use a minimum amount of space.

The philosophy of MORE

No such conservation ethic exists in the PC world. When the PC came into existence memory had become cheap, and the programming effort had moved toward high level languages such as C, PASCAL, COBOL, and FORTRAN. In these languages being byte conscious is a little more difficult and, with cheap memory prices, programmers take the attitude let them buy more memory (let them eat cake?). In the early Poly era microcomputer programmers were a cottage industry and often worked for themselves or for very small companies. In the PC era writing programs for the microcomputers was no longer a cottage industry. Large companies, often also in the business of selling hardware, dominated the programming industry. The incentive was toward larger programs, which pushed users to buy more memory.

More C

by Ralph Kenyon

After reading Bob Bybee's article "An introduction to C" in the last issue of PL, I decided to take another look at the SMALL-C compiler on the PolyGlot Library Volume 11 public domain disk. I says to myself "Let's write this for Poly's small-c." Bob's program to compute N factorial was much closer to the BASIC program than the full power of C allows. So, I will show you another version of a factorial program, and some information about small-c on the Poly.

One of the features of the C programming language is that it makes maximum use of a stack. When a subroutine is called, any arguments are pushed onto the stack first, and the stack is cleaned up afterwords. One consequence of this is that local variables are truly local; higher level variables do not get affected by procedures or subprograms which have local variables by the same name. A special case of this is when a routine calls itself (recursively); the inner calls will not affect the value of the variables at outer levels. (Of course, it is possible to bypass this feature by declaring a local variable to have null size; but, a programmer must consciously do that.)

Let's look again at Bob's program: (PL 88/3)

```
/* compute N factorial */
factorial()
  int n;
  {
  int i,f;
  f = 1;
  for (i =1, i <= n; ++i)
      f = f * i;
  return (f);
  }
```

This way of computing N factorial (written N! by mathematicians) presumes a particular definition of N factorial.

N factorial is the product of all positive integers up to and including N, and, by the way, zero factorial is one.

There is another, some say more elegant, definition of N! using the concept of recursion.

A recursive definition always has two parts. One part gives a definition for the simplest case (called the base case). The second part gives a definition for the recursive part in terms of one step simpler. Here is an example showing how N! is defined using recursion.

"N factorial" is:

Base case: When N equals zero then "N factorial" is 1.
Otherwise: N factorial is N times (N-1) factorial.

Here's another example using more mundane experiences: how to cross the road when you are walking on one side.

"Cross the road"

Base case: If no cars are coming then turn 90 degrees.
Otherwise: Walk some more and then Cross the road.

Some might argue that such a definition is "circular" (which is supposed to be bad by itself). The difference comes where the idea fits into the definition. The philosophers call the distinction the use-mention distinction, which was first characterize by the mathematician Frege. The point is that the concept is mentioned in the base case, and is not circular there. It is used in the recursive case, but in such a way that the process eventually comes to a stop. In the case of N factorial, N is reduced by one in each step, so eventually gets to 0, which is the base case.

Most BASIC's cannot do recursion, but it turns out that Poly BASIC does handle recursion. In fact, I wrote a Poly BASIC program to compute N factorial using recursion years ago. (N!.BS is on PGL-V-05.)

```
10 DIGITS 26  ON ERROR PRINT CALL(1027)
20 DEF FN N(N)
30 IF N=0 THEN Y=1 ELSE Y=N*(FN N(N-1))  RETURN Y
40 FN END
50 PRINT CHR$(12),"Demonstrating BASIC's ability to call ",
60 PRINT "functions recursivly"  PRINT "using the Factorial ",
70 PRINT "(N!) function. (by Abstract Systems, etc.)"
80 INPUT "Give me 'N' and I'll give you N! ",N
90 IF N>49 THEN PRINT " I can't count that high!"  GOTO 80
100 IF N<0 THEN 110 ELSE IF N=INT(N) THEN 120
110 PRINT " I can't compute the Gamma function!"  GOTO 80
120 PRINT N,"! =",%C,FN N(N)  GOTO 80
```

The entire definition is in lines 20 to 40, and the meat of the definition is entirely on line 30.

```
IF N=0 THEN Y=1 ELSE Y=N*(FN N(N-1))
```

Base case: IF N=0 THEN N!=1
Otherwise N!=N*(FN N(N-1))

(If the definition of N! is extended to numbers between integers and to negative numbers, the resulting function is call the Gamma function.)

Well, when I saw Bob's program, the mathematician in me jumped up and said "How inelegant! Why didn't he use the full power of C and make it recursive?" So, to satisfy the mathematician in me, here it is.

```
factorial(n)    /* mathematical notation is "N!"          */
int n;          /* our input parameter variable           */
{
        if (n == 0) return 1;
                /* base case, factorial(0) is 1           */
        else return n * factorial(--n);
                /* otherwise, recursion by the rule       */
                /* that (N)! = N * (N-1)!                 */
}       /* end factorial                                  */
```

You may have noticed that the BASIC program contains much more than just the function definition. All that extra stuff is needed to make a wholly functioning program. So, the next thing I did was try to write a program which would run under small-c for the Poly. --- Well, I had PROBLEMS! In the first place, I had never written a C program before. Although I had strung together the source files in the small-c compiler to make it a whole unit capable of compiling itself, I had not actually written anything in C itself.

Another problem I had is that I don't have the documentation for small-c. Aparently it was published in "Doctor Dobbs Journal" in May 1980, but I haven't got access to it. Does anyone out there have a copy of that issue, and would you send me a copy of the article on small-c? Well, I do have a copy of the MIX C compiler for the PC, so I looked at the documentation for that, and tried to write using its information. Unfortunately small-c doesn't have much of the MIX features, and I had to learn by trial and error what would and would not work.

Not all the data types are implemented in small-c. There is only integer, character, array types (which I think are limited to one dimension), and pointer types. The character types take up one byte of storage each, and integer and pointer types take up only two bytes. Because integers are only two bytes in size, their range is from -32767 to +32767. (One bit is for the sign and the other 15 bits is for the value of the number.)

Still, a lot can be done with small-c. By a combination of reading the small-c source code and the MIX documentation I have discovered that small-c supports the following:

```
char ...;       /* declare a character type variable */
int ...;        /* declare an integer type variable */
...[];          /* make a variable an array type */
*...            /* make a variable a pointer type */
#asm;           /* inline assembly language */
#endasm;        /* end of inline assembly language */
#include ...    /* include from another file */
#define ... ... /* define a macro /*
```

#define is limited to be a simple direct substitution macro without parameter passing as described by the MIX-C manual.

The statements implemented are: "if", "else" ,"while", "return", "break", "continue", and the special statements "{...}" (a compound statement), and ";" (the statement terminator or null statement).

The functions implemented include the assignment operator: "="; the logical operators: "|" (or), "^" (exclusive or), and "&" (and); the condition operators: "==" (equal), "!=" (not equal), "<" (less than), ">" (greater than), "<=" (less than or equal to), and ">=" (greater than or equal to); the numerical shift operators: "<<" (shift left) and ">>" (shift right); the numerical operators: "+" (addition), "-" (subtraction), "*" (multiplication),

"/" (divide), and "%" (mod); the increment and decrement operators: "++" and "--"; and finally the unary operators: "-" (negative), "*" (contents of), and "&" (address of).

And, of course, small-c implements the most important part, a new function definition and execution. The basic form for a function or procedure fits the following format:

```
name(arguments)
declare argument variables;
{
declare local variables;
body which must contain at least one return statement.
}
```

There are a few simple built in functions.
INCH(); returns a character from the keyboard.
OUTCH(c); prints c as a character on the display.
CHRDY(); returns the next waiting character, or a zero.
FOPEN(direction,unit,size,nameptr); opens a file. direction 1 is read, 2 is write, size is ignored, unit can be from 1 to 4, and nameptr is a pointer to the name of the file to be opened.

FCLOSE(unit); closes the file on unit.

FREAD(memaddr,bytes,unit); reads bytes bytes from unit to the buffer at memaddr.

FWRITE(memaddr,bytes,unit)

Also, #asm ... #endasm statements may be anywhere in the program, and the main program may also have #define, and #include statements. I imagine that #define statements can be anywhere ahead of a macro reference. #include and #define statements cannot be within function definitions. Finally, the main program must be named "main".

The small-c compiler is not very robust regarding errors. On some error conditions it hangs up. The way to run it is to not give an output file name and allow the output to be sent to the screen so you can see the errors as they are generated. Once an error free compile is obtained, then run it with an output file given. (See "Small-c Documentation" below.)

One common mistake I kept making was using "=" when I meant "==". It is not a syntax error, but funny things happen when the program is run; it does not work as expected. The other mistake I made most frequently was to mix up single and double quotes. The compiler does not tell you when you have the wrong kind, and the resulting program doesn't work correctly. I worked for a week, using the assembly language debugging tool RDB before I realized I was using the wrong syntax. The single quote converts a character to an integer. The double quote marks a string constant. The small-c compiler does not report an error when the wrong one is used.

On both the compiler and on compiled C programs which have been assembled, the stack is relocated. (If you use the SETUP.AS file included with small-c). If the program does not terminate normally, you must use the Exec command RESET to restore the stack. Otherwize, you could end up in hyperspace and bomb the system. For small programs you could leave the stack alone in the system area, but large programs may use a lot of stack, and you might risk bombing the system.

I have added to my copy of the SETUP.AS file to take care of this problem. My new SETUP file looks like this.

```
        REFS SYSTEM
        REF             ;read them all in
        ;
        ORG USER
        IDNT $,$
        ;
cold:   JMP warm
        ;
warm:   LHLD PVEC       ;Get the old CTRL-Y vector
        PUSH H          ;stash it here
        LXI H,0         ;Now let's see
        DAD SP          ;where 'here' is
        SHLD OSP        ;and save it
        LXI H,Exit      ;Our new exit vector
        SHLD PVEC       ;Now the system knows about it
        LHLD MEMTOP
        PUSH H
        LXI D,data      ;This label must be put at the
        MVI B,0         ;end of the assembled C program
zmemlp  MOV M,B         ;We're going to
        DCX H           ;zero all of memory
        MOV A,H         ;from the end of the C program
        CMP D           ;up to the top of memory
        JNZ zmemlp
        MOV A,L
        CMP E
        JNZ zmemlp
        POP H           ;MEMTOP - C programs like a BIG stack
        INX H
        SPHL            ;Set up our stack at top of memory
        CALL main       ;Here's where we call the C program
Exit    LHLD OSP        ;Get back our system stack pointer
        SPHL            ;and restore it
        POP H           ;Get our old CTRL-Y vector back
        SHLD PVEC       ;and restore it
        JMP Warm        ;now go back to the system.
        ;
OSP     DS 2            ;Here's where we save our stack
        ;
```

Well, now that we've covered all that stuff about small-c, let's see what we can do about making a real program which can be run under small-c on the Poly. Rather than go on and on with all the stuff that needed to be done, I'll just briefly describe it and then show you the whole thing.

Since we only have single character in and out functions, we need to write procedures to allow putting a string of text on the screen, to input a number from the keyboard, and to print a number on the screen. Of course, once these are written, they can be used in other C programs.

It turns out that 8! is 40,320, which is too big a number for small-c. But, that's o.k., since we're only interested in the way it's done anyway. As I believe in user-friendly (and idiot-proof) programs, I set up the beginning program administration to

tell the user what the program does, and what it's limits are. Then, the program asks the user for a number. Since I like the Poly's CTRL-X and DELETE features, I wrote those into the getnumb routine. I also wrote into the getnumb routine a provision to prevent the user from typing in an invalid number. He also knows when the number gets too big, because the routine won't take numbers that are too big. To exit the program, the user types in a number bigger than 7. A lot of the work is done in functions. Well, read the program and its documentation and you should see what I mean.

```
main()
{
int n;
        puts("This program gets a number, N, from the user,");
        OUTCH(13);      /* go to a new line     */
        puts("and computes and displays n factorial (N!).");
        OUTCH(13);OUTCH(13);    /* skip a line */
        n=0;
        while (n<8)     /* SMALL-C has small numbers    */
                {
                puts("Input a number from 0 to 7. ");
                n=getnumb();
                if (n>7) return;
                OUTCH(9);       /* do a tab     */
                puts("N = "); putnumb(n);
                puts(",  N! = "); putnumb(factorial(n));
                OUTCH(13);      /* new line     */
                }
        return;
}

factorial(n)    /* mathematical notation is "N!"        */
int n;          /* our input parameter variable         */
{
        if (n == 0) return 1;
                /* base case, factorial(0) is 1         */
        else return n * factorial(--n);
                /* otherwise, recursion by the rule     */
                /* that (N)! = N * (N-1)!               */
}       /* end factorial                                */

/* getnumb reads a positive integer from the user.      */
/* getnumb reads characters from the keyboard until a   */
/* RETURN or a non-numeric character is typed. A test   */
/* is used to detect if the number will be too big for  */
/* the compiler.                                        */

getnumb()
{
char c; /* our input character                           */
int n;  /* our result to be returned when done          */
        c=0;
        n=0;
        while(c!=13)    /* a return ends the number     */
        {c=INCH();      /* get a character              */
        if (c==24)      /* was it a CONTROL-X?          */
                {
                while (n!=0)    /* more deletes?        */
                        {       /* yes                  */
                        OUTCH(127);     /* delete one   */
                        n=n/10;         /* back up      */
                        }
                continue;       /* and loop for more    */
                }       /* wasn't a CONTROL-X.          */
        if (c==127)     /* was it a DELETE?             */
                {
                if (n!=0)       /* more deletes?        */
                        {       /* yes                  */
                        OUTCH(c);       /* delete one   */
                        n=n/10;         /* back up      */
                        }
                continue;       /* and loop for more    */
                }       /* not DELETE either            */
        if (c!=13)      /* was it a return?             */
                {
                if ((c<'0')|(c>'9'))    /* non numeric? */
                continue;       /* yes, so ignore it    */

                if ((n>3276)|((n==3276)&(c>'7')))
                                /* Going to overflow?   */
                                /* -32767 to 32767      */
                        continue;       /* yes so loop  */
        /* if we got to here, we have a valid digit     */
        /* and the number won't overflow by adding it   */
                n=n*10+c-'0';   /* compute this digit   */

                }
        OUTCH(c);       /* print either digit or RETURN */
        }               /* loop back for another while  */
        return n;       /* done, so give 'em the number */
}

/* putnumb converts an integer into positive display    */
/* characters                                           */
putnumb(n)
int n;          /* our input parameter passing variable */
{
int k[6];       /* we need an array of digits           */
int l;          /* and a loop variable                  */
        if (n==0)       /* but what if it's zero?       */
                OUTCH('0');     /* it was               */
        else    /* ok, it's not zero                    */
        {l=0;   /* zero our loop index                  */
        while (n!=0)    /* while we still have digits   */
                k[l++]=n-10*(n=n/10);
                        /* now this is a neat trick in  */
                        /* one line of code. we set     */
                        /* the lth (and while we're at  */
                        /* it, we up the index l by 1)  */
                        /* digit to the remainder digit */
                        /* (and while we're at it we    */
                        /* divide our number by 10.)    */
                        /* When the number gets to zero */
                        /* l will be set to the number  */
                        /* of digits (the number of     */
                        /* times the while loop was     */
                        /* executed.)                   */
        while (l>0)     /* while we still have digits,  */
                OUTCH (k[--l]+'0');     /* decrement    */
                        /* the digit counter and print  */
                        /* the digit (add ascii zero).  */
        }       /* end if-else. everything's printed    */
        return; /* go back to our caller                */
}       /* end of putnumb                               */

puts(str)       /* puts is like Msg (Asmb) or PRINT     */
char *str;      /* we want a pointer to the string      */
{
int k;          /* this is our string pointer index     */
        k=0;    /* start at the beginning               */
        while (str[k])  /* while getting characters     */
                OUTCH(str[k++]);
        /* display it and increment the pointer index   */
}
```

Well, that's the program. But we need to know how to compile it. So, here's the instructions from the small-c disk itself.

Small-c Documentation

by Ron Cain

Greetings! This file will describe the files which make up the small C compiler as it appeared in the May 1980 Dr. Dobb's.

All the source code for the compiler:

```
ccglobals
cc1
cc2
cc3
cc4
cc5
cc6
cc7
cc8
cc10
CCLIB
```

All but the last are C code, and were segmented into the nine files for reasons of disk space. The last is the runtime library written in assembly language.

How To Compile Small C Programs

To compile any programs written in C type:

"cc"

The compiler will ask a few questions regarding the compilation process. They were discussed in Dr. Dobb's and will be repeated below.

It will ask several questions which you can answer, namely:

"Do you wish the c-text to appear?"

If you answer "Y" followed by a return key (which I'll abbreviate <CR> from now on), the compiler will interleave the code from the original input file into the proper places in the output assembly language code. This makes reading the output file much easier but wastes disk space. I usually answer "N<CR>".

"Do you wish the globals to be defined?"

If you answer "Y<CR>" (the intended response), then all variables declared within the program being compiled will allocate room at the end of the output file. Answering "N<CR>" will suppress the allocation. I used this to split the compiler in many pieces, later join them, and not worry about multiply-defined global variables.

"Starting number for labels?"

Enter a "0<CR>". Actually, any decimal number will work. It just defines to the compiler which label number is the first to create when it begins making up label names. They take the form "ccXXXXX" where XXXXX is a number which increments each label. This option allows one to compile modules separately and then append them prior to the assembly stage without getting multiply defined label numbers. I don't use this option now, but I did once.

"Output filename?"

You choose one. Let's call the file to be created "CPROG.TX" (I usually use lower case letters when describing C files and upper case when describing Asmb files. Therefore, type "CPROG<CR>".) Not choosing one will send the output to the screen.

"Input filename?"

Enter the name for the file you wish to compile. This question will repeat each time the compiler finishes processing a file. When a bare return key is pressed (i.e. "<CR>"), the compiler will close the output file and exit to the system. This allows modules to be appended at compile time.

(3) When the compilation is complete you will have created an assembly language version of your input file. To run it, you have to assemble it. Actually, the code will generate references to routines in CCLIB, which also generates references to system routines. Therefore, you need to assemble it as follows:

(a) Create a small introductory file which you can use forever more. Let's call it SETUP.TX. It will consist of the code:

```
        REFS SYSTEM
        REF
;       ... all system REF statements ...
        ORG USER
        IDNT $,$
cold:   JMP warm
warm:   LHLD MEMTOP
        INX H
        SPHL
        CALL main
        JMP Warm
```

(b) Now assemble the 3 files:

```
SETUP
SMALLC
CCLIB
```

(4) When the assembly is accomplished, you will have an excutable version of the program. It will be complete and ready to run. In fact, all C programs were intended to be compiled this way. You can dream up more elegant ways if you like. I was forced to drop back a few paces to handle the compiler itself.

If you decide you like the compiler, feel free to change it any way you please. It was written by me and is entirely in the public domain.

I recommend you read the May and September issues of Dr. Dobb's Journal. They discuss the compiler and the directions I recommend.

Memory Management Techniques

By Bob Bybee

Memory is probably the most important resource in a computer system. There's never enough of it! Four years ago my company built a computer with 128K of memory, which we soon had to expand to 512K because the software had grown too big to fit. (The same thing happened to the original Apple Macintosh.) Now, the minimum memory we ship with a system is one megabyte of CPU memory, plus over two meg of graphics memory. One of the laws of computing seems to be, "the application will always expand to fill available memory... and then some." [An application of Parkinson's Law. -- Ed.]

The Poly is still a viable machine today, but some users find themselves severely limited by the amount of memory it can use. Let's examine the ways in which memory can be added to a computer system, and see if any of these are applicable to the Poly. Then we'll look at some ways to make the best use of available memory.

Addressing. First, think about how memory is addressed. The amount of memory that can be directly used depends upon the number of address bits the processor has. In a ridiculously simple example, think of a processor with only one address bit: that bit could only be in one of two states (off or on), and so it could only select one of two memory locations, location 0 or 1. So a one-bit machine can address (2 to the 1-th power) or two memory cells. A CPU with two address bits can address four (2 to the 2) memory cells, since those two lines can be in one of four states: 00, 01, 10, or 11. Every time you add an address bit, you double the amount of directly addressable memory. So the Poly's 8080 CPU, with its 16 address lines, can address (2 to the 16) or 65536 memory cells. (This is also called '64K' locations, with a "K" equalling 1024.) The Poly doesn't use all of that 64K for normal memory, since part of it is occupied by the video screen, ROMs, etc.

In previous issues of PL, we've noted that the Poly TwinSystem adds memory "beyond" the normal 64K limit. This is done by a method called 'bank switching'. Some extra hardware is used to switch "on" one bank of memory, while switching "off" another bank. This allows the system to have more than 64K total memory in its box, but only allows the CPU to work with 64K at a time. In the TwinSystem, two 48K boards are switched, so that only one is on at any time, and the other 16K of the system's memory is always present.

So the TwinSystem is a living example of how a Poly can have "more" than 64K, and use it well. Can we add even more memory to a Poly? Before answering, let's look at how some other computers handle their memory.

The original IBM-PC used an 8088 processor, which has 20 address bits. It can address (2 to the 20) locations, or one megabyte, or 1024K bytes of memory. Like the Poly, part of this space is dedicated for non-RAM usage: only the lower 640K contains RAM in a PC. The range from 640K to 1024K is reserved for ROMs, screen memory, etc. A technique called 'expanded memory' allows people to add more than 640K to their PC by using bank-switching in much the same way the TwinSystem does. But in this case, a small "window" (I think it's only 16K) is used to look into the entire expanded memory area (which could be many megabytes). Software selects which 16K portion of the expanded memory appears in that "window", by using the bank-select hardware.

For this technique to be useful, everyone would have to agree on things like (1) how to use the bank-select hardware; (2) what the "window" size is, or at least how to determine it; (3) where the "window" appears in the normal 1024K of address space; and (4) how to determine the size of the expanded memory. This is the subject of a standard, the Lotus-Intel-Microsoft Expanded Memory Standard. You may see PC literature referring to the LIM EMS standard, and many vendors now make boards that follow it. With such a standard, it's possible for many software vendors to write programs that can "break" the 640K barrier. Lotus was one of the authors of this standard, since their spreadsheet product, Lotus 1-2-3, is a memory gobbler.

The IBM-PC's 8088 has some younger relatives which have more address bits. The 80286 has 24 and the 80386 has 32, allowing those CPU chips to directly address 16 megabytes and 4 gigabytes respectively. But in their valiant effort to be "IBM-PC compatible", machines built using these newer CPUs must live within the 640K limit imposed by the original PC's architecture. Imagine having an address space of 4 gigabytes (4,096 megabytes or 4,194,304 kilobytes) and only being able to use 640 kilobytes of that!

Well, the computers which use the '286 and '386 have something called 'extended memory'. Don't confuse this with expanded memory, described above. Extended memory is memory that exists outside the 1 megabyte space that an 8088 can access. Of course, a program that's written for the IBM-PC has to live within the 1 megabyte limit (640K of RAM, actually), and so it can't use extended memory at all. But the operating system of a 286/386-based machine, or a program specifically designed to run on a 286/386 (and not on a PC), can use this memory. It's often used for non-program functions, such as a RAM-disk, print spooler buffer, or to hold extra programs which can be called upon with certain keystrokes. (See "TSR programs", below.)

Another use of extended memory is multi-tasking. When you want to run more than one program at a time, and each one is an MS-DOS program for the PC, each one wants to live within the lowest 640K of RAM. Obviously they can't all do this. But a crude way to perform multi-tasking is to copy each program into the lowest 640K, let it run for a while, then copy it out to extended memory and copy some other program in for a while. In this way, the programs share the CPU, since they take turns running. But this is an inefficient way to multi-task, since it involves copying large programs around in memory, many times per second. "Real" multi-tasking systems don't eat quiche, and they don't waste their time

copying programs either.

Idea Number One for the Poly:
A New CPU with More Address Space.
[The IEEE 696 Concept. -- Ed]

There are a couple of 8080-compatible CPU chips available which will run Poly programs, but also have more address bits available. The Hitachi 64180 can access 512K bytes, and the Zilog Z280 can access 16 meg. Building a new CPU board with either of these chips (and about 512K of RAM) would give the Poly greater memory capability, plus make the system run faster. And, without a great deal of effort, we could use this memory in some of the same ways the 286/386 world uses its extended memory: print spoolers, RAM-disks, etc. A program like Poly Peripherals' PCALC spreadsheet could even use it to expand the amount of data storage available. But we couldn't use it to run BASIC or assembly language programs that are larger than the Poly's 56K RAM limit, since the Poly operating system and BASIC interpreter only know about using 56K at a time. It would be very difficult to modify them.

There's also a CPU chip called the V20, which can be plugged into a PC in place of its 8088 CPU. The V20 acts just like the 8088, except it's a bit faster; but it also can emulate the instruction set of the 8080. Unfortunately that's not enough to let us run Poly programs on a V20-equipped PC. The Poly does things like directly writing to screen memory, which would not exist in the proper memory locations on a PC. That's the main problem I've run into in trying to use the V20 as a Poly-emulator on the PC. If anyone has a simple way around this problem, I'd like to hear about it! [The simplest way around this that I can think of is to create a time-share routine which watches the Poly screen ram area and updates the PC screen whenever the Poly screen ram changes. -- Ed.]

Bank Switching To The Extreme. Returning to the idea of adding memory by bank-switching: suppose we build a memory board that works much like the "expanded memory" scheme, with a window through which we can see some larger memory. Only this board has a very small window, and only lets you see one byte at a time! Ridiculous, you say? Actually, that's the way some RAM-disk boards work. These have also been called solid-state disks, since they can't really be accessed like normal RAM. Their "window", and the hardware that controls what byte it "looks" at, are not RAM locations at all, but I/O port locations. So they can only be used by programs that know how to manipulate those I/O ports to store and retrieve data. As the term RAM-disk implies, they are designed for emulating a disk, where sequential, one-byte-at-a-time access is fine.

Idea Number Two for the Poly:
Use a Ram-disk for More than a Disk.

Any data can be stored in a RAM-disk. Two or more programs can even share the storage in a RAM-disk card, if they keep out of each other's way. A two-megabyte RAM-disk might be used for one 512K disk emulator, a 512K print spooler buffer, and a 1 megabyte data area for a spreadsheet program. The only thing you can't do easily with a "windowed" RAM board is run programs out of it.

Virtual Memory. Large mainframe computer systems have used virtual memory for years. Virtual memory is a way for the system to pretend that it has more memory than it really does, by using disk space as a substitute for CPU memory. Suppose a CPU only has 64K of real RAM, but it has several megabytes of disk space out there somewhere. When a program starts running, it uses "real" memory; but when it tries to go outside the range of "real" memory that exists, the virtual memory hardware catches that fact. It notifies the operating system (OS) that a non-real address (a virtual address) is being accessed. The OS then decides what part of "real" memory hasn't been used lately, and writes that part of memory out to disk. That memory is now available for use in a different part of the address space, and the virtual memory hardware re-maps memory so that it appears where the running program wants it.

This is a very simplified description, but it tries to capture the essence of what a VM system does. So far, there are no microcomputer systems which use virtual memory, since VM requires a great deal of extra hardware, larger disks, and a complex operating system. Sorry, guys, we won't see a Poly/VM system anytime soon. But some programs written for the Poly do use certain techniques borrowed from VM. The Adventure program, written by Bob Martin for the Poly, consists mainly of a file with text messages in it. This file is much larger than 64K, so it won't fit into the Poly's memory. But Bob puts part of the file into memory at one time, moving other parts into memory as they are needed during the game. He even named the file VM.VM, reflecting the virtual-memory nature of the program's organization.

You could say that any program whose data is kept partially on disk is a VM program, but that's being too generous. Effective VM systems will "intelligently" move pages of data into and out of memory as needed, sometimes anticipating what data is likely to be needed next and reading it into memory too. This is more efficient than simply reading/writing records of data as they are needed, but also more difficult to implement.

TSR Programs.

Anyone who reads a PC magazine will encounter something called a TSR, for Terminate and Stay-Resident program. In a system without true multi-tasking, TSRs are a way to load multiple programs into memory at once and keep tham all handy. On the PC, a TSR is usually activated with some magic combination of keystrokes, like ALT/SHIFT. TSRs can act as simple keyboard filters, or they can completely take over the system and run a new program for a while, perhaps bringing up a calculator program or notepad for temporary use. When you're finished with a TSR, it will typically disappear, leaving you back in whatever program you were previously running.

Well, gee, doesn't that some familiar? The Poly has had TSR programs for years, only we didn't know

what they were called. Any program which loads itself into high RAM and is then available to perform functions on command, can be called a TSR. This includes the simple "hit Ctrl-U to print screen" utility presented in PolyLetter almost 10 years ago, as well as the complex "hot-key" program described by Ralph recently. [The following are Poly "TSR" programs: Clock.GO, Inhibit.GO, blink.GO, Fence.GO, print.GO, Cursor.GO, B-EDIT.GO, edit.GO, PramKey.GO, Submit.GO, KISS.RL, Poly's RDB.GO, and Gnomus.GO (See: "Poly's Best Kept Secret" in PolyLetter 8706.) -- Ed.] The Poly can use TSRs just as easily as the PC or any other system, with the exception that it doesn't have as much memory to hold them, or as many different keys that can invoke them.

Device driver programs are similar to TSRs in some respects. They don't perform any particular function at the press of a keystroke, but they DO load themselves into high RAM and remain there to perform functions at a later date. In both the Poly and IBM systems, a device driver (or a TSR) must load itself into unused memory, then pass some information to the operating system so the OS knows to leave it alone, and not re-use that memory area for another program later. [The various versions of Driver.DD for the Volume Manager are examples. -- Ed.]

On the Poly, the only way to load a TSR program has been to put it into high memory, just under MEMTOP (except for a few very tiny programs which can fit into a small area of unused RAM around address 0C80). Since not all Poly systems have the same amount of RAM, the procedure is:

1. Load the program into RAM at 3200 hex, like all normal programs do.

2. Look at MEMTOP, which indicates the last available RAM location.

3. Move the program code to just underneath MEMTOP, and "relocate" the program so it can run properly at that address.

4. Adjust MEMTOP so the next program knows that less RAM is now free.

Step 3 was always the tough one, until Poly came out with its (still undocumented) RELOC feature of the assembler. [I understand that RELOC is documented in the latest (or next?) version of the System Programmers Guide. -- Ed.] A program can be written with RELOC and then relocated to run at any address. I wrote utility programs MAKEREL and LOAD which assist in this process; Ralph Kenyon offers similar programs MakeRel and LoadRel.

Calling Assembly From BASIC. The TSR concept can be used to add assembly-language routines to a BASIC program. Sometimes you want to perform a function in BASIC that can only be done in assembly. Or perhaps you just want to speed up part of a BASIC program by rewriting it in assembly. The 'CALL' statement allows a BASIC program to call an assembly language subroutine. This is a powerful feature of Poly BASIC.

But where do you put the assembly routine that BASIC calls? A common way is to make it a TSR. This means it must be loaded into memory before running the BASIC program, because BASIC uses memory just under MEMTOP (and works backwards, down toward zero) when loading your BASIC program. Once you load the BASIC interpreter and your BASIC program, memory looks like this:

```
MEMTOP--------------
 BASIC program
 loads here
  .
  .
--------------------
free memory
--------------------
BASIC interpreter
3200H---------------
```

The "free memory" area is used for your BASIC program's variables, arrays, string manipulations, and so on.

So, we can put an assembly language routine in high memory, just like a TSR, and then call it from BASIC. Several successful programs do that: Form.OV and KISS.RL are both assembly routines which provide useful functions for BASIC programs, like screen editing of input. But the drawback of this approach is that you have to remember to load the TSR before starting BASIC. And if you exit your BASIC program, the TSR is still occupying high memory, doing nothing useful, unless there's a way to "de-install" it. [All of Abstract System's TSR programs can be de-installed by the command sequence: ENABLE, ZAP, START. --- Ed.]

A new approach for BASIC/assembly calls. I developed a new way to call assembly routines from BASIC. This method is used in Poly Peripherals' PCALC spreadsheet. The idea is: instead of loading the assembly routine into high memory, load it into the "free memory" in BASIC's memory map. But first you need a place in free memory that you're sure BASIC won't be using. PCALC does this by DIMensioning an array in memory, large enough to hold the assembly language program, then loading the assembly language program into that array and relocating it to run at that address. The address of the array can be obtained from BASIC's MEM function. PCALC passes that address to an overlay, Pcld.OV, which performs the load-and-relocate function. Then the assembly routines in that array can be used by BASIC's CALL statement. [This sure beats hand coding the routines in BASIC statements! -- Ed.]

An array may seem like a peculiar place to store a subroutine, but with relocatable code, it doesn't matter. A relocatable routine can be loaded anywhere! And this method has the advantage that the memory is automatically "de-allocated" when you exit BASIC. No high RAM is used, MEMTOP need never be changed, and the only program affected by the loading of the assembly routine is the BASIC program that loaded it. If you make changes to the BASIC program, the size of it will generally change; this changes the address of each array and variable used

by the program. [Array addresses are determined by the version of BASIC, the size of DIGITS, and the number of variables previously declared or referenced, not by the size of the program. Although, the size of the program often affects the number of variables declared or used before an array is declared. -- Ed.] But again, that doesn't matter, since the assembly routine is loaded into the proper area each time the BASIC program runs.

I'll make the Pcld.OV overlay available to anyone who sends me a diskette (5" SSSD) and return postage.

In summary. Several techniques can be used to increase the amount of memory available on a Poly. Memory beyond 64K can be added through I/O-based bank-switching, as in a RAM-disk. Or it can be built into a new CPU card, using a chip like the Z280 which can directly address more RAM. Either way, most Poly programs won't be able to make use of it, since BASIC and Exec aren't designed to handle extra memory space. That space could be useful for other things, though, such as RAM-disk or print-spooler areas, or data areas for programs that need a lot of data storage. TSR programs and subroutines might also go into these "extended" memory areas.

I invite Poly users to write or call me, with their thoughts on these or other subjects!

[Bob Bybee can be reached at: Poly Peripherals, 5011 Brougham Court, Stone Mountain, GA 30087, (404) 498-3556 -- Ed.]

How to UNSAVEP

by Ralph Kenyon

CONTINUED from PolyLetter 88/3...

You recall I told you that BASIC keeps a flag to remember if a program was protected? Fortunately, there is a way to find that flag. The first thing to do is create a trivial BASIC program, call it BP, "10 PRINT "OK",INP(1)" which will keep running. Second, save a copy of it in SAVEF mode, called BPF. Next, save a copy of it in SAVEP mode, called BPP. Now, we want to ZAP memory and execute BPF. When the "OK" is printed we push the LOAD button. Next we use the SAVE command to SAVE a few sectors of memory contents from the end of BASIC up into the data area.

BASIC C04 is 52 sectors long. That is 34 sectors in HEX. Each 4 sectors is 1K, so that is 13K, or 0D K in Hex. If we take 3200 and add D00 to it and we get 3F00. This is the first sector in memory past where BASIC.GO loads. So, we answer the SAVE command's questions FROM, LOAD, and START with 3F00. (This is for BASIC C04; other BASICs might be different and must be computed separately.)

For the unprotected version I call the file BDF. Next, I go through the same steps for the protected version, only I call the resulting file BDP. Now that I have two copies of part of BASIC's data area, for the same program, and with the same starting conditions (namely, memory ZAPPED, and the program auto-started), I can run the program COMPARE.GO to see what bytes are different. Make a list of the addresses of the bytes that are different, (perhaps by selecting the output to the printer option on compare).

Next comes the tedious part. We must execute the protected program, push the load button, GET BASIC, ENABLE the system, call up the front panel with CTRL-Z, and L(oad) the address of the byte to be tested. Then we enter a "0" to turn the byte off, hit G to exit from the front panel, and then use the REENTER command to get back into BASIC. Then we simply type LIST. If we found the flag, we get a listing. Otherwise, we get "I can't do that to a protected file!", or get dumped back to Exec.

Sounds like a lot of work? It can be, depending on how far out the flag is. But, once you have found the flag with this technique, you can use it to decrypt any program for that version of BASIC. However, would you believe, there is an easier way? And without fooling around with the protect flag. But, as you readers of mysteries know this is the perfect time for another one of those cliff-hangers. TO BE CONTINUED...

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

---

1. MAXELL diskettes: 5-1/4" 10 hard sector -- $15 per box.
2. HayesSys modem software (for the Micromodem 100) $25.
3. (A:S) Spell, a good spelling checker for $35.
4. Abstract Systems Exec (Enhancements & bugs corrected) $35.
5. Abstract Systems Proms (Enhancements & bugs corrected) $35.
6. PolyGlot Library Volumes: $6 each.

(Send $1.00 for a complete catalog--(free with any order).)
(Make check or money order payable to Ralph Kenyon.)

Readers Requests

Readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. How would CP/M be of use. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Hardware update recommendations, Source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

Bit Bucket

Anything to save typing....

Did you know that Edit can reopen the output file on any drive or in any subdirectory by using the wild card? When I want to move a file I am going to change to another subdirectory, I hit ESC,CTRL-O just as if I was opening a new output file. Then I type in the new drive and subdirectory name followed by "<*" (example: "<3<NPL<*") and hit return. Edit will use the old (input) file name! (The extension will be that used for the output file.)

In This Issue

Coming Soon

Modems and Communications software; More: BASIC for Beginners, how to UNSAVEP protected Programs, System Programmers Notes, Converting Poly BASIC to PC BASIC, Help, BugNotes, Public Domain Software, etc.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL

Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter, please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter* are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Overseas $20.00 yr., payable in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).

PolyLetter

Editorial

Well, my printer died, and I had to rush it out to be repaired. It seems to be working well now, except that they must have changed the fonts a little. My old PolyLetter setup files didn't work and I had to find new width values by trial and error. I think the intercharacter spacing in proportional is a little bit more now.

Letters

Dear PolyLetter, September 20, 1988

I wrote the C factorial routine the way I did to make it recognizable to BASIC users, not to neglect the recursive features of C. After teaching a course in programming, I discovered that recursion is a difficult concept for beginners to master, and I didn't want to cloud my C article with that concept.

I found your small-C article very interesting. I purchased a book on small-C which is (as far as I know) still available through most computer bookstores, and should explain some of the things you had questions about. You may also want to obtain the original "bible" on C, the white book with the big blue C on its cover, which is the language definition (at least until ANSI got ahold of it). That book also defines standard I/O functions such as printf and scanf, which you may want to write for small-C if you intend to do much more with it. They would allow you to do things like

```
printf("Here are several lines\n\n");
```

instead of

```
puts("Here are several lines");
OUTCH(13); OUTCH(13);
```

Bob Bybee, Stone Mountain, GA

Dear PolyLetter, September 26, 1988

Regarding PL 88/3... enjoyed Bob Bybee's Introduction to C. I tried the public domain version you sent a while back, and while it can be made to work, the user interface is a little primitive. Anyone done anything about that? On a slightly different tack, if anyone out there has ported the James E. Hendrix version to the Poly, his Small-c Handbook is an excellent tutorial and self-study guide for compiler construction. Maybe PolyLetter could serve as a clearing house for that compiler in our disk format if there is enough interest. There seems to be quite a few small-c compilers around, but few have such a comprehensive reference to accompany them.

Regarding PL 88/4... One idea mentioned in Bob Bybee's article about Memory Management that keeps resurfacing from time to time is RAM-DISK. John J Warkentin wrote a pair of articles in PolyLetter 8503 and 8504 about interfacing Digital Research Computers LS-100 RAM Disk to the Poly. I don't know if the LS-100 is still in production, but you might refer readers to this as a good hardware article. I have no experience with this product, but perhaps John could be persuaded to write a short follow-up for PolyLetter. Keep up the good work! --- Allen Daubendiek, Palm Bay, FL.

Dear PolyLetter, October 6, 1988

One thing that sounds interesting is a comment made by Don Haywood in the 8802 issue. He says that nearly every IC can be replaced by a CMOS chip. Can this be done piecemeal, or does it have to be done all at once? Are parts replacement list available? For maintaining your equipment in good running order in out of the way places like up here this could be good information to have. Could you please get more information on this subject? -- Percy Roy, Edmonton, Alberta, Canada.

[Perhaps our readers could send in information to answer Your question? What documents tell what chips can be substituted for which? I only know that a 74LS367 can be usually be substituted for an 8T97. Anyone know of other substitutions for chips on the Poly cards? -- Ed.]

Dear PolyLetter

The Poly is and will continue to be a very useful computer. Please include

application notes showing how Poly is being used (feature reader applications); continue with program examples (BASIC, Exec). Print more extensive BASIC or Assembly Programs Advertise any applicable hardware/upgrades - Graphics boxes, bank-switched memory, keyboards, monitor, printers, hard disk subsystems, floppees, float-point mother boards, etc... -- Kenneth Lowe, West Valley City, UT.

How to UNSAVEP

by Ralph Kenyon

CONTINUED from PolyLetter 88/4...

Now, finally, here's the secret easy way to UNSAVEP encrypted programs. You must do this on a disk without Vmgr, and with NO INITIAL file. You need to know how big your program is in sectors, and what your Top of RAM is.

First use the DISPLAY command to find out what Top of RAM is. write down the 4 digit hex number. Most likely it will be FFFF or DFFF, but it could be anything. Next, look at the directory and see how big the SAVEP program is. Next, convert the size to hexadecimal. Remember the size, but tack a '00' onto the end of it. For example, if the file size was 43 sectors, converting to hexadecimal gives us 2B and tacking on 00 gives us 2B00. Next subtract this from memtop. Let's use DFFF. DFFF - 2B00 gives B4FF. Now, add 1 to this. B4FF + 1 gives B500.

Now, we are going to run our program and push the load button. Then we are going to SAVE from address B500, Start address 0403, Load address B500, 2B sectors. File Name MI.GO

Next we do a ZAP followed by MI, followed by GET BASIC, followed by REENTER and we have it.

Of course, for BASIC C04, it's even easier. Run the program, Press LOAD, GET BASIC, REENTER, and you have it.

TICKLER.BS

People have asked for more extensive BASIC programs. Here is one I use every day. It is a simple suspense/tickler program which I use to remind me of things I have to do. In this article I will explain what the program does and show how I have converted it to run under GWBASIC (IBM).

I put the invocation as the last command in my INITIAL file. Then the program looks in the data file and reminds me of anything that is overdue, or coming due in the next week. I called the program "simple" because it does not maintain the tickler data file. I must do that with Edit. In designing the data file of tickler items, I decided to put a date code in front of each item. The date code has six digits in the format YYMMDD so that it can be easily sorted in the due date order. I also limit the total length of the line to one screen line - 64 character.

What does TICKLER.BS do? Well, the first thing it does is to setup variable storage space. Next, it reads the date from a file named "Date.DT" which other system programs maintain. [See the JAN/FEB 81 PolyLetter.] The date in this file is going to determine whether a tickler item is past due, current, or still pending. TICKLER.BS converts the date from Date.DT into the form YYMMDD for comparison with tickler items. This is stored in the variable D0$. The tickler also computes two other dates. One is the next day "tomorrow", and the cutoff date, one week in the future. Tomorrow's date is stored in D1$ and the cutoff date is stored in D2$.

Next, the tickler opens the datafile TICKLER.TX (which I happen to keep in subdirectory "DT"). It then begins rummaging through the file and processes one line at a time. The processing is done in a subroutine which starts at line 150. The first check is to see if the date is more than a week in the future; if so the item is skipped.

Since I originally wrote this system I decided it would be nice to be able to be reminded of things which have no deadline. So I allowed it to recognize a date code of "000000" and just give me a "To do:" reminder. Lines 170-200 determine the status of the current item and display that information. Lines 210 & 220 display the due date and line 230 displays the content portion of the tickler item. Variable L9 counts the number of printed lines so the program can stop after a page full.

There are two user defined functions in TICKLER.BS. One, "FN D$", is designed to figure out what day of the week the date corresponds to, taking into consideration leap years. Remember that leap years occur every 4 years except for century years not divisible by 400. (The year 1900 was not a leap year, but the year 2000 will be.) Line 630 computes whether the current year is a leap year. Poly's BASIC sets a logical test to 1 so L is set to 1 if either the year is divisible by four and not divisible by 100 or the year is divisible by 400. Line 640 adds the value of L if we are already past February. In line 660 the number of leap days since the year 0 is computed and added in. Actually, the calendar doesn't go back that far but it computes the day of the week correctly.

We would ordinarily need to add a "fudge factor" to get the day of the week right but I did it by shifting the array D9$ until it matched up right. Now when the number of days since the year zero is divided by 7 and 1 added to the remainder, it picks out the correct day of the week.

The second function, "FN S$" converts a space in a string to a numerical zero. "88 2 5" becomes "880205". This is to match the numbers in the file.

Lines 420 to 550 hard code items which depend only on the day of the week. You can change them to anything you want to be reminded of for that particular day of the week. Note that in the days I have used there is "L9=L9+1", which counts the printed line.

The heart of the program is in lines 560 to 600. Line 570 reads a line and tests for the end of the file. Line 580 processes the line and tests to see if I typed the ESCAPE key after the page was displayed. Line 600 returns to Exec without the Exec sign-on.

One note. Line 220 has the century year "19" "hard coded" in, and would need to be changed to "20" when we get to the year 2000. I expect my Poly will still be running then.

Program Listing

```
10 DIM M0$(12:3),D0$(1:20),D1$(1:6),D2$(1:8) \MAT READ M0$
20 DATA "JAN","FEB","MAR","APR","MAY","JUN"
30 DATA "JUL","AUG","SEP","OCT","NOV","DEC"
40 DIM I1$(1:64),D9$(7:9),D6$(1:9) \MAT READ D9$
50 DATA "Saturday","Sunday","Monday"
60 DATA "Tuesday","Wednesday","Thursday","Friday"
70 DIM M9(12),D7$(1:6),D8$(1:8) \MAT READ M9
80 DATA 31,28,31,30,31,30,31,31,30,31,30,31
90 POKE 11656,0 \REM Turn off command file mode
100 F9=0 \FILE:4,OPEN,"Date.DT",INPUT
110 INPUT:4,D0$ \FILE:4,CLOSE \GOTO 270
120 IF ASC(D0$,1)=32 THEN D0$=RIGHT$(D0$,LEN(D0$)-1) \GOTO 120
130 IF ASC(D0$,1)<>32 THEN D0$=RIGHT$(D0$,LEN(D0$)-1) \GOTO 130
140 RETURN
150 IF LEFT$(I1$,6)>D2$ THEN RETURN
160 IF LEFT$(I1$,6)="000000" THEN PRINT "To do: ", \GOTO 230
170 IF LEFT$(I1$,6)<D0$ THEN PRINT "Past due! ",
180 IF LEFT$(I1$,6)=D0$ THEN PRINT "Today: ",
190 IF LEFT$(I1$,6)=D1$ THEN PRINT "Tomorrow: ",
200 IF LEFT$(I1$,6)>D1$ THEN PRINT "Scheduled: ",
210 PRINT FN D$(MID$(I1$,1,6)),", ",M0$(VAL(MID$(I1$,3,4))),
220 PRINT (VAL(MID$(I1$,5,6))),", 19",MID$(I1$,1,2) \L9=L9+1
230 L9=L9+1 \PRINT TAB(7),MID$(I1$,7,64) \F9=1
240 IF L9>13 THEN L9=1 \PRINT "Waiting...", \Z=INP(1)
250 PRINT CHR$(24), \RETURN
260 INPUT "What is the date? (MONTH DAY, YEAR) :",D0$
270 FOR I=MEM(D0$) TO MEM(D0$)+2
280 POKE I,PEEK(I) AND 95 \NEXT \REM Fold
290 I=0 \FOR I=1 TO 12 \IF LEFT$(D0$,3)=M0$(I) THEN M0=I
300 NEXT \IF I=0 THEN 260 ELSE GOSUB 130 \D0=VAL(D0$)
310 GOSUB 120 \Y0=MOD(VAL(D0$),100)
320 D0$=FN S$(STR$(Y0,%#2I)+STR$(M0,%#2I)+STR$(D0,%#2I))
330 D0=D0+1 \IF D0>M9(M0) THEN D0=D0-M9(M0) \M0=M0+1
340 IF M0=13 THEN M0=1 \Y0=Y0+1
350 D1$=FN S$(STR$(Y0,%#2I)+STR$(M0,%#2I)+STR$(D0,%#2I))
360 D0=D0+7 \IF D0>M9(M0) THEN D0=D0-M9(M0) \M0=M0+1
370 IF M0=13 THEN M0=1 \Y0=Y0+1
380 D2$=FN S$(STR$(Y0,%#2I)+STR$(M0,%#2I)+STR$(D0,%#2I))
390 D6$=FN D$(D0$) \PRINT TAB(15),"Today is ",D6$, \L9=1
400 PRINT ", ",M0$(VAL(MID$(D0$,3,4))),
410 PRINT (VAL(MID$(D0$,5,6))),", 19",MID$(D0$,1,2),"."
420 IF D6$<>"Monday" THEN 440
430 L9=L9+1 \PRINT "     Do the laundry."
440 IF D6$<>"Tuesday" THEN 460
450 REM Tuesday stuff
460 IF D6$<>"Wednesday" THEN 480
470 REM Wednesday stuff
480 IF D6$<>"Thursday" THEN 500
490 REM Thursday stuff
500 IF D6$<>"Friday" THEN 520
510 REM Friday stuff
520 IF D6$<>"Saturday" THEN 540
530 L9=L9+1 \PRINT "     Take the trash to the dump."
540 IF D6$<>"Sunday" THEN 560
550 L9=L9+1 \PRINT "     Water the plants."
560 FILE:4,OPEN,"<?<DT<TICKLER.TX",INPUT
570 INPUT:4,I1$ \IF I1$="" THEN 590
580 GOSUB 150 \IF Z=27 THEN 600 ELSE 570
590 IF F9=0 THEN PRINT "No tickler items."
600 Z=CALL(0,";")
610 DEF FN D$(D7$)
620 D1=VAL(D7$) \Y=INT(D1/10000) \D1=D1-Y*10000-100 \Y=1900+Y
630 L=(Y/4=INT(Y/4) AND Y/100<>INT(Y/100)) OR Y/400=INT(Y/400)
640 D2=Y*365 \IF D1>200 THEN D2=D2+L
650 FOR I=1 TO 12 \IF D1>100 THEN D1=D1-100 \D2=D2+M9(I)
660 NEXT \D2=D2+D1+INT((Y-1)/4)-INT((Y-1)/100)+INT((Y-1)/400)
670 D2=D2-INT(D2/7)*7+1 \RETURN D9$(D2) \FN END
680 DEF FN S$(D8$)
690 FOR I=0 TO 7
700 IF PEEK(MEM(D8$)+I)=32 THEN POKE MEM(D8$)+I,48
710 NEXT \RETURN D8$ \FN END
```

IBM VERSION

Converting this to GWBASIC required quite a number of changes. Some changes I already knew about. [See The Other BASIC, PL 88/2, 88/3.] It turns out that the FILE statement I described in 88/2 doesn't work. I must have been trying some other BASIC when I wrote that. Anyway, I ported the program over to the clone using PROCOMM with an ASCII download and the PRINT command on the Poly. Then I just tried to RUN the program in GWBASIC. When I got errors, I'd change the program in the Poly so I could use the Poly editor, and port it over again. After 4 hours and dozens of trials, I finally got it right.

The first two changes required are substituting a colon for a backslash and substituting a semi-colon for a comma in print statements. These are the only changes required to lines 160, 170, 180, 190, 200, 330, 340, 360, 370, 430, 530, 550, 580, 640, 650, and 660.

The next thing I had to change is the DIM statements. GWBASIC just uses DIM statements only for arrays. Another thing

is that GWBASIC doesn't know about MAT statements. I had to convert each one to a FOR-TO-NEXT loop. These changes affect lines 10, 40 and 70.

Another difference is in how the MID$ function works. In Poly BASIC MID$(A$,A,B) represents the portion of A$ starting with character position A and including character position B. In GWBASIC MID$(A$,A,B) represents the portion of A$ starting with character position A and including B characters. These changes affect lines 210, 220, 230, 400, 410

I mentioned the functions above. It turns out that GWBASIC only allows one line functions of the form

DEF FN NAME(parameters) = EXPRESSION

It does not understand multiline functions. To deal with this, both multi-line functions had to be converted to subroutines. In both cases the input parameter must be set, then the routine called with a GOSUB, and then the output parameter must be used. In the case of FN D$, which affects lines 210 and 380-390 the value of D1 is set and then the subroutine at line 620, which used to be the function, is called with GOSUB 620. The subroutine at 620 sets the value of D2, which is used as an index into D9$ to pick out the day of the week.

FN S$ was used to build the date. Since M0, D0, and Y0 are already defined, we only need for the routine to return its result in a string variable. D8$ does the trick. Lines 320, 350, and 380 are affected by these changes.

Both the functions themselves also had to be changed. In the case of FN D$ which becomes the subroutine at 620, lines 610, 620, 640, and 670 had to be changed. The change in line 640 is subtle. It turned out that in GWBASIC a logical test returns the value -1. Consequently, I had to SUBTRACT L instead of adding it.

In the case of FN S$, the changes are more extensive. While GWBASIC does have PEEK and POKE statements, and they work the same, it does not have the MEM function; the whole concept of FN S$ needed to be changed. Statements 680 to 770 are the result. Notice that a MID$ function appears on the left of an equals sign in an assignment statement. Unlike Poly BASIC, GWBASIC allows assigning to a portion of a string.

Statement 90 turned off command file mode so that the wait function worked properly at the end of each page. Without statement 90 the program would get a zero byte from the command file and keep right on going. In the clone, I have no idea where the batch file status byte is, and even if I did know where it was, it wouldn't make a difference since batch files don't get program inputs from the file anyway. So we just get rid of line 90.

GWBASIC has a special variable which has the system date in it. That is DATE$ and its format is mm/dd/yyyy. We don't need the lines to read the file, so we can delete lines 100, and 110 (we still need the GOTO); we also don't need line 260 since DATE$ is always there and the program will never have to ask the user. Lines 270 and 280 as well as the subroutine from lines 120 thru 140 are involved in converting the date from the file format to the internal format so also aren't needed. Since we know the format in DATE$, the conversion process is much simpler as is shown in the changes to lines 290 to 310.

In Poly BASIC INP(1) waits for a character from the keyboard and returns it. GWBASIC can't do this. Here we must use the INKEY$ variable and test it to see if it's null. This affects line 240. Line 240 is also affected by the lines per page on the screen. The Poly has 16 lines and the clone has 25 lines. I can't use CTRL-X to wipe out the line on the clone screen either. Since I haven't figured out how to do it, I just deleted this nice little message altogether resulting in changes to line 250 as well.

The file access statement in line 560 is completely different. The Poly returns a null string when the end of file is reached so the end of the tickler file is tested for by seeing if the input variable is null. That doesn't work in GWBASIC. However, GWBASIC does have an explicit End Of File test statement. Of course and EOF test must be made before reading from the file, so this statement comes before reading. Statement 570 is the result.

Finally, the way to return to DOS from within a GWBASIC program is different from returning to Exec from within a Poly BASIC program. There are various ways from Poly BASIC to return to Exec, but I have only learned one for GWBASIC. Line 600 has the result.

There is one subtlty I found out that bears noting. GWBASIC doesn't like to read commas into a string and won't eat one when it's in the data file. It assumes that the comma marks the end of the string and the start of the next one. So, If you use this program on your IBM (compatible) don't put any commas in the line of data, or you will get weird results

Well, now that I have gone over all the

changes, here is the entire program as it would list in GWBASIC. I have left the "empty" statements in so the line numbers would coincide with the corresponding Poly BASIC line numbers. To completely finish the program requires deleting these statements and renumbering. Well, here's the listing:

TICKLER.BAS

```
10 DIM M0$(12) :FOR I=1 TO 12 :READ M0$(I) :NEXT
20 DATA "JAN","FEB","MAR","APR","MAY","JUN"
30 DATA "JUL","AUG","SEP","OCT","NOV","DEC"
40 DIM D9$(7) :FOR I=1 TO 7 :READ D9$(I) :NEXT
50 DATA "Saturday","Sunday","Monday"
60 DATA "Tuesday","Wednesday","Thursday","Friday"
70 DIM M9(12) :FOR I=1 TO 12 :READ M9(I) :NEXT
80 DATA 31,28,31,30,31,30,31,31,30,31,30,31
90 REM
100 F9=0
110 GOTO 270
120 REM
130 REM
140 REM
150 IF LEFT$(I1$,6)>D2$ THEN RETURN
160 IF LEFT$(I1$,6)="000000" THEN PRINT "To do:"; :GOTO 230
170 IF LEFT$(I1$,6)<D0$ THEN PRINT "Past due! ";
180 IF LEFT$(I1$,6)=D0$ THEN PRINT "Today: ";
190 IF LEFT$(I1$,6)=D1$ THEN PRINT "Tomorrow: ";
200 IF LEFT$(I1$,6)>D1$ THEN PRINT "Scheduled: ";
205 D1=VAL(MID$(I1$,1,6)) :GOSUB 620
210 PRINT D9$(D2);", ";M0$(VAL(MID$(I1$,3,2)));
220 PRINT (VAL(MID$(I1$,5,2)));", 19";MID$(I1$,1,2) :L9=L9+1
230 L9=L9+1 :PRINT TAB(7);MID$(I1$,7,58) :F9=1
240 IF L9<24 THEN RETURN
242 Z$=INKEY$ : IF Z$="" THEN 242 ELSE Z=ASC(Z$)
250 L9=1 :RETURN
260 REM
270 REM
280 REM
290 M0=VAL(MID$(DATE$,1,2))
300 D0=VAL(MID$(DATE$,4,2))
310 Y0=VAL(MID$(DATE$,9,2))
320 GOSUB 680 : D0$=D8$
330 D0=D0+1 :IF D0>M9(M0) THEN D0=D0-M9(M0) :M0=M0+1
340 IF M0=13 THEN M0=1 :Y0=Y0+1
350 GOSUB 680 : D1$=D8$
360 D0=D0+7 :IF D0>M9(M0) THEN D0=D0-M9(M0) :M0=M0+1
370 IF M0=13 THEN M0=1 :Y0=Y0+1
380 GOSUB 680 : D2$=D8$ :D1=VAL(D0$) :GOSUB 620
390 D6$=D9$(D2) :PRINT TAB(15);"Today is ";D6$; :L9=1
400 PRINT ", ";M0$(VAL(MID$(D0$,3,2)));
410 PRINT (VAL(MID$(D0$,5,2)));", 19";MID$(D0$,1,2);"."
420 IF D6$<>"Monday" THEN 440
430 L9=L9+1 :PRINT "     Do the laundry."
440 IF D6$<>"Tuesday" THEN 460
450 REM Tuesday stuff
460 IF D6$<>"Wednesday" THEN 480
470 REM Wednesday stuff
480 IF D6$<>"Thursday" THEN 500
490 REM Thursday stuff
500 IF D6$<>"Friday" THEN 520
510 REM Friday stuff
520 IF D6$<>"Saturday" THEN 540
530 L9=L9+1 :PRINT "     Take the trash to the dump."
540 IF D6$<>"Sunday" THEN 560
550 L9=L9+1 :PRINT "     Water the plants."
560 OPEN "TICKLER.TXT" FOR INPUT AS #1
570 IF EOF(1) THEN 590 ELSE INPUT #1,I1$
580 GOSUB 150 :IF Z=27 THEN 600 ELSE 570
590 IF F9=0 THEN PRINT "No tickler items."
600 SYSTEM
610 REM
620 Y=INT(D1/10000) :D1=D1-Y*10000-100 :Y=1900+Y
630 L=(Y/4=INT(Y/4) AND Y/100<>INT(Y/100)) OR Y/400=INT(Y/400)
640 D2=Y*365 :IF D1>200 THEN D2=D2-L
650 FOR I=1 TO 12 :IF D1>100 THEN D1=D1-100 :D2=D2+M9(I)
660 NEXT :D2=D2+D1+INT((Y-1)/4)-INT((Y-1)/100)+INT((Y-1)/400)
670 D2=D2-INT(D2/7)*7+1 :D7$=D9$(D2) :RETURN
680 M1$=STR$(M0)
690 IF LEN(M1$)=3 THEN M1$=RIGHT$(M1$,2)
700 IF LEFT$(M1$,1)=" " THEN MID$(M1$,1,1)="0"
710 D3$=STR$(D0)
720 IF LEN(D3$)=3 THEN D3$=RIGHT$(D3$,2)
730 IF LEFT$(D3$,1)=" " THEN MID$(D3$,1,1)="0"
740 Y0$=STR$(Y0)
750 IF LEN(Y0$)=3 THEN Y0$=RIGHT$(Y0$,2)
760 IF LEFT$(Y0$,1)=" " THEN MID$(Y0$,1,1)="0"
770 D8$=Y0$+M1$+D3$ :RETURN
```

Here is a sample data file.

```
000000Call mom
881201Pay Mortgage
881210Send holiday cards
890101Pay Mortgage
890120Sales Tax return due
890201Pay Mortgage
890214Valentines Day
890301Pay Mortgage
890401Pay Mortgage
890415Income tax due
890501Pay Mortgage
890601Pay Mortgage
890701Pay Mortgage
890801Pay Mortgage
890901Pay Mortgage
891001Pay Mortgage
891101Pay Mortgage
```

Anyone who wants this program can send me a disk and $1 for shipping, or send me $3 for the program on a fresh disk in either Poly or PC format or $5 for both formats. I'll include this sample data file.

Two Poly TSR's
Gnomus with edit

These two programs have changed my life! Their combined effect is making life so much easier that I simply have to share the result with you.

To review, Gnomus is a TSR (Terminate and Stay Resident), hot key program which gives the user the power of Edit's ESCAPE definition library macro facility while in Exec or any other program. I will talk about my version of Gnomus, which has been derived from the Poly version which Poly was good enough to place in the public domain (PolyGlot Library Volume number 25). Also, edit is another TSR program which allows editing any line on the screen and sending the result to the keyboard buffer

to be executed.

First, how I have changed Gnomus. Poly's Edit program uses the ESCape key to signal that a defined macro is to be executed, or that a new definition is to be entered. The most common use is to insert format commands into the text. For example Poly's TXdef.ED file contains 'p' defined as '{par}'. However, a macro can contain control characters also. To insert a control character into a macro definition, one precedes the character by a carrot ('^'). The left arrow key is the same as CTRL-T and that would be inserted into a macro definition by typing '^T'. I have defined 'H' as '{he}{end}^T^T^T^T^T'. When I type ESC,H I get '{he}■{end}' with the cursor in the middle so I am ready to start typing the actual header. I don't have to move the cursor back in between the '{he}' and the '{end}' as it is already there.

Macro definitions can also execute other macros. To insert an ESC into a macro definition one uses '^['. I often use the escape colon sequence to strip extra blanks at the end of lines. The sequence is to start at the beginning and look for a space followed by a carriage return, back up past the carriage return, delete the space, and back up one more so that another space-return can be found. The sequence looks like this:ESC:^F ^M^[^T^D^TESC. To insert that into a definition one must convert each ESC into '^['. Suppose I define the character 9 with this sequence. I would also need to insert another carrot to prevent the escape that marks the end of find from being also the end of the colon sequence. That gives me

ESC=9^[:^F ^M^^[^T^D^T^[ESC

as my input sequence. So whenever I press 'ESC 9' this little routine strips out any trailing blanks. Let's see how Gnomus is different.

To prevent conflict with Edit, I changed the activation key for Gnomus to the backspace key. That key is almost never used in the Poly. But just in case someone would want to get a backspace character into the system, Gnomus looks at the second character. And if it is a backspace, Gnomus sends that one character through to the system. Two backspaces become one.

If the character was not a backspace, Gnomus checks for certain special cases. The equals character ('=') signals the start of a new definition. The CTRL-X character signals that the internal macro buffer is to be cleared. (Wer're going to start fresh.) As with Edit, Gnomus can load and save macro definition files. The CTRL-L character signals that a macro definition file is to be Loaded and asks for the file name. The CTRL-W character signals that a macro definition file is to be Written to disk and asks for the file name. Also because a delete character cannot be put into a Gnomus definition, I have set it up so that the backspace character is converted into a delete character.

Gnomus does not have the continuous repeat facility (escape colon) sequence that Edit has, and another Gnomus macro cannot be executed from within a Gnomus macro. But I have found a way around that using edit and will get to that a bit later. But first, here are some of the common simple Gnomus macros I use. For each of the numbers 1 thru 7 I have a routine defined that sets the wild card path to that drive number, clears the screen and lists that drive using the wild card path.

```
1 = ^X# 1^M^L^XI #^M
2 = ^X# 2^M^L^XI #^M
3 = ^X# 3^M^L^XI #^M
4 = ^X# 4^M^L^XI #^M
5 = ^X# 5^M^L^XI #^M
6 = ^X# 6^M^L^XI #^M
7 = ^X# 7^M^L^XI #^M
```

The first '^X' clears anything that might be in the command buffer that I might have started to type. '# 1^M' sets the wild card path to 1; the '^M' is the carriage return that executes the command. This is followed by '^L', which is a CTRL-L and clears the screen, and another CTRL-X which clears the CTRL-L out of the command buffer. Finally, the 'I #^M' lists '#' which has just been redefined. What I see is the screen clear and the drive being listed in response to two key strokes. Here are some other routines I commonly use.

```
S = u<Szap^M            Invoke SuperZap
Z = ZAP^MST^M   ZAP memory and START the program in high RAM
g = c<Call<Gini^M       Execute command file to Dial Gini
p = PAG^M               Page the printer
s = RL<Submit           Prepare to execute a SB file.
λ = ^L^XI $^M           List the default path
c = ^L^X                Clear the screen
```

These routines are all pretty tame compared to some of the ones that use edit.

First, a review of edit. edit is another TSR program which loads itself up into high memory and waits for the proper activation key sequence. edit is invoked when one types a CTRL-DELETE character. Since my printer uses the CTRL-DELETE character to set 17.5 characters per inch, I sometimes want to get that character into the system. Like Gnomus, edit looks at the second character to see if it is also a CTRL-DELETE. Two CTRL-DELETES make one. If the second character is NOT a

CTRL-DELETE, edit throws the character away and invokes the editor function. The cursor jumps up to just behind the last character on the line above where it was before (but it won't go off the screen).

edit supports a lot of the same features that Edit does. CTRL-B takes one to the end of the top line; CTRL-E takes one to the end of the bottom line; CTRL-X deletes everything to the left of the cursor; CTRL-W deletes one word to the left of the cursor; and the arrows work somewhat like Edit. Left and right arrows work the same. Up and down arrows move to the end of the line above or below. ESC up and down arrows move to the head or tail of the current line. ESC [and ESC] move left or right one word. ESC left or right arrow inserts a block marker arrow just like in Edit. ESC CTRL-C copies the marked block to the cursor location. edit allows inserting a control character by preceding it with a carrot; '^X' is converted into the CTRL-X character (which looks like the greek capital omega.) To get a carrot into edit type it twice. Two carrots become one.

Now that I have reviewed the basics of edit, we can look at some Gnomus definitions that assume that one is in edit. For example, I list a drive using one of the BACKSPACE number gnomus definitions. The I hit ESC to exit from the LIST function [{A;S} Exec only]. Now I have a listing of files on the screen. I hit CTRL-DELETE and a space to invoke the edit. Then I use the arrow keys to move up behind the file name I want to do something with. At this point I am in edit and have the cursor on a file name. The wild card is also defined to the drive listed.

Suppose I want to TYPE or PRINT the file; I have gnomus definitions which will do the job. First it needs to skip to the front of the file name, clear out the rest of the line (to the left), insert the command and a space, and finally to exit the editor giving the line to Exec as a command. It sounds complicated, but it is just what one would have to do by hand using only edit. Here are the Gnomus definitions for TYPE and PRINT

```
P = ^[[^XPR <#<^M
T = ^[[^X^^L^^XT <#<^M
```

First, an explanation of BACKSPACE P. The first three characters, '^[[', get converted by Gnomus into ESC [, which is the edit command to move to the left of the current word. The next two characters, '^X', get converted by Gnomus into CTRL-X which is the edit command to clear the line to the left. The next two characters, '^^', get converted by Gnomus into a single carrot, '^'. edit takes the single carrot and the L (^L) and converts it into the CTRL-L character which it inserts in the line. A similar sequence happens with the '^^X'. Gnomus converts it into '^X' and edit converts it into a CTRL-X character which it inserts into the line. Next the characters 'T <#<' are inserted, which will be in front of the file name. Finally the ^M character is converted by Gnomus into a CTRL-M which is a carriage return. This signals edit to stuff the entire line into the keyboard buffer. So, the keyboard buffer gets a CTRL-L, a CTRL-X, the characters 'T <#<', the file name and extension just as it apeared on the screen, and a carriage return. When Exec processes this line it results in clearing the screen, clearing the command buffer, and then getting and executing the new command, to type the file.

I did this with two key strokes to invoke the editor, a couple of up arrows to move up to the right file, and two more key strokes to invoke the Gnomus command T. It sure saves a lot of typing.

The other Gnomus definitions I use which assume a listing on the screen and the editor invoked are P for PRINT, shown above, and:

```
D = ^[[^X^^XDE <#<^M          Delete the file
E = ^[[^X^^XED <#<^M          Edit the file
L = ^[[^X^^L^^XL <#<^M        List the subdirectory
N = ^[[^X^^Xu<New <#<^M       Flip the New bit
X = ^[[^X^^X<#<^M             Execute the file
f = ^[[^Xformat <#<^M         format the file
% = RL<Dio35^M                Connect 48tpi Driver
```

The above all end in a CTRL-M and automatically execute. The following do not execute immediately as I am expected to add some information to finish the command.

```
A = ^[[^Xu<ARISE <#<^[^R      Undelete a file
C = ^[[^Xu<Scopy <#<^[^R <    Copy a file
R = ^[[^XREN <#<^[^R <#<      Rename a file
m = ^[[^Xu<Move <#<^[^R <
```

ARISE expects a number to say which deleted version to get. Scopy expects a new output file name. RENAME expects a new file name.

Now, let's get fancy. I said I had found a way around the fact that Gnomus definitions cannot include other Gnomus commands. I do it by setting up the commands in edit and the getting out of Gnomus to let edit call the commands.

The first example is a bit simple. It starts out by invoking edit, clearing a line, inserting the Gnomus invocation to clear the buffer, inserting the Gnomus invocation to Load a new Gnomus definition file, and then invoking another Gnomus definition (loaded with the other file).

G = ^_ ^X^^H^^X^^H^^LDT<PI^^M^^H^^J^M

Here's a blow by blow description. '^_' gets converted by Gnomus into the CTRL-DELETE character which causes edit to be evoked. The space following '_' is the throw-away character that edit needs. '^X' is converted by Gnomus into CTRL-X which tells edit to clear the line to the left (in this case everything on the line to give us a clean line to work with). Gnomus next converts '^^' into '^' so edit gets '^H' which it takes as an instruction to insert a CTRL-H which is the backspace character. The same thing happens again with the '^^X' being converted into a CTRL-X character. At this point the line being edited has a backspace character and the CTRL-X character. Similarly the next six characters, '^^H^^L' get converted into BACKSPACE CTRL-L to be added to the line. Now we get some simple characters passed to edit by gnomus and 'DT<PI' gets addded to the line being edited. As before, Gnomus converts '^^M' into '^M' and edit converts that into a CTRL-M which gets added to the line being edited. Another '^^H' becomes a BACKSPACE and '^^J' gets converted into CTRL-J which is the LINE-FEED character.

So we have on the line being edited the characters BACKSPACE, CTRL-X, BACKSPACE, CTRL-L, the text 'DT<PI', and the characters CTRL-M,BACKSPACE,LINE-FEED. The final '^M^' gets converted by Gnomus to CTRL-M which is a RETURN, and that signals edit to send the line to the keyboard buffer. When Exec begins processing the characters from the buffer, the first BACKSPACE,CTRL-X is intercepted by Gnomus and Gnomus clears its internal macro buffer. The second BACKSPACE, CTRL-L is also intercepted by Gnomus and it then takes the characters 'DT<PI' and the carriage return as its instruction to load the Gnomus definition library file named PI.Gn from subdirectory DT. The final BACKSPACE, LINE-FEED is intercepted by Gnomus as a command to execute the Gnomus macro named by the LINE-FEEF character. That particular Gnomus file is the file of definitions I use when I am working on PolyLetter articles and the LINE-FEED key is defined to set the path to the directory where the PolyLetter articles are and to list that directory.

Wow! If that's a bit simple, how could we get more complicated? Well, I have three definitions which are designed to help maintain Gnomus macro definitions. The first one is not too complicated. It invokes the editor, clears a line, deletes the current macro file, and instructs Gnomus to write out the new version. This makes saving a changed definition super-easy.

u = ^_ ^X^^XDE DT<Gn^^M^^H^^WDT<Gn^M

[the '^X' clears edits working line and the '^^X' insures that the command buffer gets cleaned out before the delete command is processed. This prevents bombing from some stray character being in the command buffer.

The other two are used for changing definitions themselves. Remember that the BACKSPACE, QUESTION sequence display the current macro definitions on the screen? Well, two of these definitions work with that display.

9 = ^[^Q^H=^S^^[^^M; ^M5
e = ^[^Q^S^S^S^Suu^T^^H=^[^R^^[^M

The first, '9', is for deleting a definition. Gnomus converts '^[^Q' into the edit command ESC up-arrow which sends the cursor to the beginning of the line. '^H' is converted by Gnomus into the BACKSPACE character. At this point the BACKSPACE character and an '=' character have been inserted in the beginning of the line (in front of the macro character name). Gnomus converts the next two characters, '^S' into the edit command CTRL-S, which is a right arrow, and edit sends the cursor just to the right of the macro name character. The next three characters, '^^[' get converted by Gnomus into '^[' which edit converts into the ESCAPE character. Next, the '^^M' similarly gets converted into a CTRL-M or RETURN character. Finally, a semicolon and a space are inserted before the rest of the stuff on the line. The final '^M' is converted by Gnomus into CTRL-M, or RETURN, and signals edit to send the entire line to the keyboard buffer. But, look what is on the line. There is a BACKSPACE, followed by an EQUALS, followed by the macro name character, followed by an ESCAPE followed by a RETURN, which is the Gnomus definition sequence for defining a null macro. This wipes out the definition. I remember nine as nein or 'no'. The rest of the line gets executed by Exec as a comment.

Now we can get a little fancier (a LITTLE he says). The second definition is designed for updating a definition which is on the screen, and has been changed by using edit.

e = ^[^Q^S^S^S^Suu^T^^H=^[^R^^[^M

Its instructions are to go to the beginning of the line, move right 4 spaces, delete the three characters ' = ', move left one, insert a BACKSPACE and an EQUALS, move to the right end of the line, insert an ESCAPE, and terminate. This feeds a revised definition to Gnomus. The backspace character is converted to a DELETE character by Gnomus.

Now you will see why I said 'a LITTLE fancy'. After using these two programs for a while I decided I wanted to be able to add paths to the wild card without having to type it all over again. Usually the path is a directory which I have listed on the screen. The idea seemed not to difficult, but implementing it got fancy. Here's the basic idea. I use the DISPLAY command to put the current wild card path on the screen. I use edit and Gnomus to put the file name together with the old wild card path and define it as the new wild card path. The trick involved writing a macro that would not only start within edit, but exit from edit and then reevoke it! I also found that I needed to do some screen cleanup to prevent extra junk from getting into the path. Well, her's the definition in two line.

```
# = ^R^R^X^Q^Q^[[^X^^XDISP^^M^^_ ^^Q
<^^[[^^X# ^^[^^R^[^R^^^^M^^^^L^^^^XL #^M
```

The first three backspaces convert to DELETs and remove the extension (the '.DX') from the sub-directory name. Next we drop down two lines and clean (erase) the line Then we move back up, go in front of the sub-directory name, clear the line, insert a CTRL-X (to prevent stray character bombing), the command DISP, and a RETURN. Next we insert characters to evoke the editor again, and insert instructions for the editor to do the following: Move up behind the existing path, insert a '<', move in front of the path, clear the line, insert '# ' which becomes the command to set the new path, move to the end of the line (where the added path name will go). Then we go to end of the current line, which puts us behind the name we want added, and issue commands to cause edit to insert a RETURN, a CTRL-L, and the LIST command.

When we finaly execute this macro it results in the wild card path being redefined to add the file name to the existing path and then clears the screen and lists the new subdirectory. Now that's what I call fancy. This definition is too long to be changed in edit, so I had to make a command file to insert it.

You must pardon my enthusiasm, but use of these two programs has completely changed how I use the Poly. Why, I have been learning about Hypercard lately, and I realized that the Poly, with Gnomus and edit can do much of what Hypercard can do.

For example, I started creating a file which has philosophy definitions in them. In each file I have words in upper case which have their own definitions. Each file is one screen size (like most of my help files) and has an imbedded form feed to clear the screen. To browse around like in Hypercard, I set the path and type one of these files. Then I hit CTRL-DELETE to invoke edit and move the cursor to the word whose definition I want to see. A BACKSPACE T pops the new file onto the screen. It wouldn't be too difficult to have tailored Gnomus files which would execute some program and display a graphic on the screen along with the text file typed.

Hypercard indeed. Poly can do it too!

I get $15 for each of Gnomus and edit by themselves but mention this article and I'll send both along with my Gnomus macro definition file, and the help files for only $25. (shipping included).

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

PolyMorphic Systems
7334-H Hollister Avenue,
Santa Barbara, CA 93117

One source for new and reconditioned systems, hard disk sub-systems, parts, conversions, manuals, and service. We've got it all! CALL (805) 685-6238.

FOR SALE: 400 4116 memory chips. $200 for the lot. Charles Steinhauser, 2213 Voorhees, Apt. 102, Redondo Beach, CA 90278, Phone 1-213-214-0326.

FOR SALE: Poly 8810 box with power supply and mother board. $50 plus shipping. Charles A. Thompson, 2909 Rosedale Avenue, Dallas, Texas 75205-1532, Phone: (214)-368-8223

For Sale: 8813 Twin system with MS and 10M Priam HD, 8813 DSDD 5" with 10M Priam HD, 3 VTI boards, 1 SD controller, Poly Keyboard & Screen enclosure, 3 Serial cards, 1 Cassette interface, 1 Disk Controller Tester, 20 Boxes 8" disks, Field Service Manual, Dealer's kits, Manuals, etc. $1,500 for the lot. - Charles Trayser, 415-651-5931.

Abstract Systems, etc.
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

DISKS - MODEMS - PROMS - SOFTWARE - SPELL

1. MAXELL diskettes: 5-1/4" 10 hard sector -- $15 per box.
2. HayesSys modem software (for the Micromodem 100) $25.
3. [A;S] Spell, a good spelling checker for $35.
4. Abstract Systems Exec (Enhancements & bugs corrected) $35.
5. Abstract Systems Proms (Enhancements & bugs corrected) $35.
6. PolyGlot Library Volumes: $6 each.

(Send $1.00 for a complete catalog--[free with any order].)
(Make check or money order payable to Ralph Kenyon.)

Readers Requests

Readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. How would CP/M be of use. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Also wanted are hardware update recommendations, source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

Bit Bucket

Did you hear the one about the fellow who was spreading bread crumbs along the border of his property? His neighbor came out and asked: "Nasruden, why on earth are you doing that?". "I'm keeping the tigers away.", answered Nasruden. His neighbor protested: "But, there aren't any tigers within hundreds of miles of here!" "Effective, isn't it?" replied Nasruden.

In This Issue

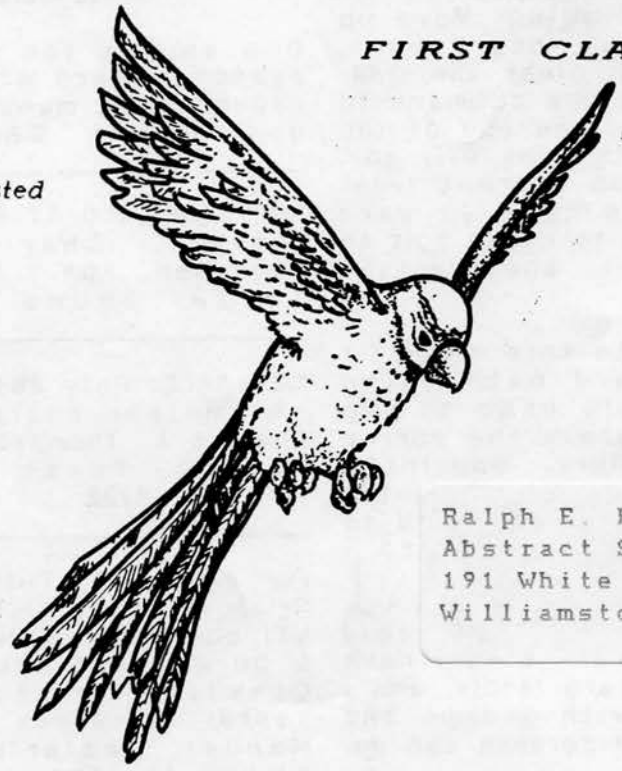
Coming Soon

Modems and Communications software; More: BASIC for Beginners, System Programmers Notes, Help, BugNotes, Public Domain Software, etc.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL



Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter; please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter*nital) *are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Oveseas $20.00 yr., payabh% in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).*

PolyLetter

Poly Closing!

Sirous Parsaei, the current owner and president of PolyMorphic Systems called at the end of November to tell me that he has decided to close PolyMorphic Systems. Sirous tells me that he still has 5 rooms of parts, and that if people needs any thing they should call him. (Prices negotiable.)

Sirous reports that he expects to vacate the facilities by the end of January 1989. He did say that he could continue to provide some repair of boards after that time, but he would no longer have parts and anyone getting service would need to provide their own parts. For that purpose Mr. Parsaei provided his home phone number. The number is (805)-967-4423.

Until that time, he can still be reached on the company phone.

PolyMorphic Systems
7334-H Hollister Avenue,
Santa Barbara, CA 93117
CALL (805) 685-6238.

Do expect to get an answer machine. Sirous says there is no message on the answer machine, so when you hear the beep just leave a phone number for him to get back to you. He expects to be liquidating the assets in the meantime.

Editorial

Well, the other shoe has finally dropped. Poly is closing. We have expected this day for years, but have always hoped that it would be tomorrow rather than today. Alice would like Poly to close every other day. Close yesterday or close tomorrow, but never close today.

Poly has had a long history in a business where companies often don't last more than a couple of years. There were lots of S-100 companies back in Poly's heyday. Everyone knows of MITS and IMSAII. And North Star lasted for quite a bit of time. Remember Vector Graphics? That was the pretty green machine with the white racing stripe. The company's entire top management were all women.

Of course, the S-100 concept was a myth for these companies. It's true that they all used the S-100 bus (sort of), and many different boards would work in many different machines. The S-100 concept eventually became the IEEE-696 standard. But, that standard differed from all the old S-100 machines. Each of the old S-100 machines had something unique that prevented it from getting away from its parent company. In the Poly, we have a proprietary operating system, a weird clock speed, and a unique disk format.

Proprietary characteristics prevent compatibility. Well, some people have stripped out the Poly boards and inserted IEEE-696 boards, but most have remained loyal to the Poly with all its idiosyncrasies. It is still a good system. For my money, it has the best operating system going even today. With my fast 96 tpi drives and now a hard disk, I have enough computing power and storage space to meet most of my need for some time to come. And with six spare Polys around, I'll probly be computing on the Poly for years to come.

Even now, the Poly with its 1.8432 MHz 8080 CPU does many things faster than my 8 MHz NEC-V20 CPU on the XT clone. I delight in telling the story of my experience with WordPerfect. I created a cross-reference list of files and sub-directory on the XT-clone and tried to SORT it. SORT reported that the file was too big to fit in memory, so I went into WordPerfect and created a MACRO to remove the date and time stamp on the file. This way I expected to shrink the file enough to sort it. The macro was defined to mark the beginning of a block, move over past the date and time, mark the end of the block, delete the block and move down one line. I placed the cursor at the beginning of the date and told WordPerfect to execute the macro 500 times. (I had at least that many lines.) Well, 29 MINUTES later it finally finished.

There's gotta be a faster way!

I shipped the file over to the Poly

using PROCOMM on the clone and DownLoad on the Poly. Then I went in with the Poly editor and used an ESCAPE colon sequence. Find a carriage return, move over so many characters, and delete so many characters. The Poly did this is under two minutes!!!

To be perfectly fair to WordPerfect, I went back and tried the trick again. This time I defined the macro as a temporary macro and had it do almost what I do on the Poly. It finds C:, backs up 4 spaces, deletes the date time info and moves right 4 spaces. Executing this temporary macro 2000 times took about 6 minutes. I can't explain the difference in time. Can you?

Well, back to Poly. I know of no one using a Poly-88 these days. Does anyone know of one in active use? Well, the Mother company and the eldest child are gone, but the other children live on amongst us. Let us hope the last four spots remain open for years to come.

```
     ***********
   *             *
  *    R I P    *
 *               *
 *  PolyMorphic  *
 *    Systems    *  ********
 *  (1976-1989)  *  * Poly *
 *               *  *  88  *
 *****************  ********

........ ........ ........ ........
. 8813 . . 8810 . . 88MS . . 88HD .
........ ........ ........ ........
```

Moving Subdirectories

by Ralph Kenyon

One annoying property of the program DIRCOPY.GO is that it copies all the files first and then copies subdirectories. The problem with this is that Gfid has to go way out on the disk to get the subdirectory. This adds to the lookup time.

One way to avoid this is to create all subdirectories first and then run DIRCOPY. I have done this by creating a dummy file, X.TX, with only one space in it and then COPYing it from subdirectory to subdirectory until all the desired subdirectories are created. Then I delete all copies of the file and pack the disk. (I usually create a command file to do the job.)

Now when I use DIRCOPY, all the subdirectories are already created and at the begining of the disk, so lookup time is saved. This is easy to do on 5-1/4 inch drives as there cannot be too much data on the disk to copy. On larger disks, however the process may take quite a bit of time. DIRCOPY is much slower than IMAGE.

I have devised a method for moving a subdirectory to the beginning of the disk using the system resources and some extra programs. The process is not difficult, but one mistake can destroy the whole directory! (This process won't work for system disks.)

This is not for the weak hearted!

The first thing that must be done is to create some disk space at the beginning of the disk. To do this I find out which file or files are near the head of the disk and then edit or copy them to another location. Once that is done, some free space is created into which to copy a new directory. We must have 4 free sectors, but more won't be a problem because the extra sectors will be recovered during PACK.

Here's how I make sure I get the right space. First I call up my Print to a File program. Let's say I name the file SL (for sorted list). Then I set the Printer to LOG all transactions. Now I run my slist program on the drive; this program gives a sorted list of all files on a drive cross referenced to subdirectory. To make sure I get the disk address, I do this in the ENABLEd mode. (slist knows about ENABLE and DISABLE just like the system LIST command.) Once slist has listed all the files, I select Printer Screen which causes the output file to be closed and stops LOGging. Then I edit SL to clean up some junk. The printer driver inserts form-feeds after each page, so these must be gotten rid of. This I do with an ESC colon sequence.

ESC : ^ F ^ L ^ [^ D ESC

(The spaces are not typed; they are just for readability here.) This means: repeat until done, find a form-feed, and delete it.

Also, after each page a colon and a delete character must be removed. So I go back to the beginning and use another ESC colon sequence. This one is

ESC : CTRL-F ^ M : ^ [→ CTRL-D CTRL-D ESC.
■ ɳ ^M : ^[ν ϵ ϵ ■

The second line shows what it looks like on the screen before pressing the final ESC. It can also be done by using the carrot and the character. Since a right arrow cursor movement is the same as CTRL-S this would result in:

■^F^M:^[^S^D^D■

This means repeat until done, find a

carriage return (CTRL-M) followed by a colon, move right one space (past the delete character [looks like a cursor]), and delete the delete character and the colon. There is usually some garbage at the end of the file and a line of junk at the beginning of the file which must also be deleted. These I do manually. Once I have completed this cleanup I have a file which only has listing entries. Now I can exit from the editor.

Now that I have cleaned up file SL which contains a listing of the files sorted in file name order, I need to re-SORT the file into the order of the disk address. This I do with the SORT program. I instruct SORT to sort starting in column 6, which is the start of the disk address in enabled mode listing. I tell SORT to read from the file SL and output to the file SD. The command sequence is:

SORT /+6 SL SD

Once this is done, I can DELETE file SL and then take a look at file SD.

File SD now has a listing of all files on the disk in the same order that they are stored on the disk, regardless of what sub-directory they appear in. By looking in SD, I can find the first files that are NOT subdirectory files. If the first file that is not a directory is 4 sectors or longer, I need only move it, but if the first file is less than 4 sectors, I may have to move more than one file.

Moving the file

If the file is a text file the task is simple. I just EDIT it and then exit from the editor without changing it. EDIT will rewrite the file in a new location and delete the original. If the file is not a text file, then I COPY it with a new name, delete the old one, and RENAME the new copy back to the original name.

At this point, I repeat the process above to obtain a new listing of files on the disk sorted in disk address order. This new listing should show the deleted file or files with at least four sectors of deleted space. I do this just to be safe.

Now that there is deleted file space large enough to hold a directory, we need to get a new name entered in the directory. I make a note of the disk address of the deleted file. Let's say it turns out to be '1C'. The next thing I need to do is to list the directory I wish to enter the new file name in and see what the first file that starts after this address is. Just to make things easy here I will assume the file name is NX.GO. What I am going to do is rename the file name with a name big enough to be broken into two separate file names. To do this we need to know what information is kept in a directory entry.

Each file directory entry consists of the following:

1. A one byte flag.

This tells how big the file name is. (This byte also has three flag bits to identify DELETEd, System, or New files, but these won't affect us since the file will not be a system file, it won't be deleted, and since we have renamed it, the new bit will be cleared.)

2. 1 to 31 bytes for the file name. (How many is determined by the flag byte.)
3. Two byte file extension.
4. Two byte disk address.
5. Two byte file size.
6. Two byte Load address.
7. Two byte Start address.

In the case of a two letter file named 'NN.DX' this format would be fNNDXddssLLSS. Now, If I am going to rename the old filename 'NX' I must add this many bytes to the name. The old file name fNXGOddssLLSS will have the new file name added to the front of it (I want the new name before the old NX file). So the new director entry must look like this:

fNNDXddssLLSSfNXGOddssLLSS

But, part of that is data which is alread included in the file directory entry, which I mark below with dashes.

fNNDXddssLLSSfNXGOddssLLSS
-NNNNNNNNNNNNNNEX--------

So, I need to rename NX.GO to NNDXddssLLSSfNX.GO Now I have a directory entry big enough to break into two two-letter file names and make two directory entries. This part must be done in SuperZap. Remember the disk address I mentioned above? (1C) That is the only thing we need to know before we go into Szap.

I ENABLE the system and call up Szap. Next I select the drive with the : command (colon, drive, RETURN). If I am entering the file in the main directory, we will be there already. Touch ESCape and the ASCII text will be displayed on the right side. Using the cursor keys, I find the newly named file name. The file name is 15 characters long, so the existing flag byte will be '0F'. I move the cursor to the flag position and press '2' and the space bar. The '0F' will change to '02', and the cursor will be over the first character of the name. By pressing the right arrow 4 times the cursor will be over the first 'd' character ('64'). This gets changed to the low byte of the disk address (1C in this example); I press '1' followed by a 'C' and

then the space bar. Since the high byte is zero, I then press '0' followed by another space. This sets the disk address to '1C'. Since all directories are 4 sectors long, I now press '4' followed by a space and another '0' followed by a space. After this the cursor is now over the first 'L'. Since all directories have 101 as their LOAD and Start addresses, I just press '1' and the space bar four times. At this point the cursor will be over the 'f' ('66') character. This byte will be the new flag byte for the old file name. I change this to a 2 by pressing '2' and the space bar. Now, If I am working in the main directory I just need to press CTRL-E and Szap will do the rest. If, however, I am working in a sub directory, I need to back up to the first sector of the directory and press CTRL-C so Szap will compute the new subdirectory checksum. The CTRL-E gets me out of Szap and back to Exec. At this point, if everything has gone right, I can list the director and see a new file named NN.DX just before the file named NX.GO. The first time you try this please use a backed up disk!!

The final task that must be accomplished is to clean up the new sub-directory, since it still has the garbage left there by deleting the old files. For this purpose I use Szap and CLEAN or SCOPY. With Szap I select the disk address of the directory, /1C in this example, and then use Szap's Z command to zero everything in the four sectors of the directory. After exiting from Szap, the directory isn't correct yet because it doesn't all the proper data. But not to worry, the program CLEAN will fix things up.

CLEAN <d<NN.DX

will clean up the directory and set things up properly.

As an alternative, I can copy from any empty subdirector onto NN.DX using SCOPY and the * option. I usually have a spare directory called SD.DX.

SCOPY SD.DX NN.DX *

will copy SD onto DX. Since SD was correctly formatted as an empty directory, NN.DX will get 'fixed up' by the copy process.

The final step remaining is to pack the disk to get rid of the deleted file names that point at the same location as the new subdirectory, and to recover any extra sectors (if, for example the deleted files had added up to 5 or more sectors).

I use this same process to "move" an existing subdirectory to the front of a disk. I use SCOPY to copy the old subdirectory on top of the new subdirectory; since the files will be duplicated, I delete the old subdirectory, or delete all files in it, or CLEAN it before packing the disk.

To Boycott or Not to Boycott

by Bob Todd

Reprinted from Amateur Computer Group of New Jersey News Volume 13, No. 2, February 1988.

In the nineteen sixties the responsibilities of corporations and the way they deal with the public started to gain attention. If a firm was guilty of excessive pollution, this came to be seen as grounds for choosing not to do business with that company. More recently, the concept has been extended to cover a wider range of business activities. For example, many states and municipalities have enacted prohibitions against doing business with firms that operate in South Africa.

Similarly, individuals don't hesitate to withhold their trade from a local grocer of pharmacist that jerks them around, uses unfair advertising tactics like bait and switch, or is just unpleasant to deal with.

In the personal computer industry we've seen many cases of buyers switching sources. As an example, copy protection proved to be particularly troublesome for many firms. Some, which had a very large percentage of a market, found that competitive products which featured no copy protection swiftly penetrated the market they had controlled. In fact, some governmental agencies forbade the use of copy protected software because the copy protection installation program usually made unapproved changes to the operating system - a potentially serious breach of the security of the data on the system.

For my part, I won't use copy protected software on my system. I have legitimate, licensed versions of programs, but I don't install them until the copy protection is broken. I never allow a program to fiddle with hidden files or file allocation tables on my system. Thank goodness this kind of problem is going away.

Now, let's look at a new problem that's rearing its ugly head:

Look and Feel Lawsuits

Would the PC industry be what it is today if IBM had sued everyone who made a PC compatible machine? I doubt it.

But note what IBM did do. If a compatible maker merely copied the IBM's software ROMs, IBM sued for copyright

infringement. This issue had been settled in the Apple-Franklin lawsuit. Otherwise, they merely reviewed the competitor's code and said whether or not it was "too close" to theirs. Responsible vendors made an arrangement with IBM to conduct this review before their product went to market. A few did not, and the resulting lawsuit either put them out of business or forced them to modify their software.

Note also, that when PCs became a commodity market, IBM did its usual thing and refused to compete. They changed the ground rules and came out with a new product: the PS line of computers. This is typical IBM practice. When IBM wants to eliminate the competition, they move on and force the competition to spend development money to keep up.

This is not, however, the only approach to eliminating the competition. There is another darker side.

Perhaps the most important single element in the extraordinary success of microcomputers was spreadsheet software. When Apples running Visicalc appeared on the desks of CEO's and CFO's around the country, it scared the dickens out of IBM executives. The PC was born to counteract this invasion of what IBM thought of as their sacred territory: business computing in the Fortune 500 community.

Spreadsheets were quickly adapted for the IBM-PC and its emerging competitors. Visicalc, Supercalc and Lotus fought it out to become the premier corporate spreadsheet program. With a large number of new features, with graphics, with fast presentation and reporting modes, Lotus quickly moved to the forefront. They had what was perceived to be the superior product.

Now a curious thing has happened.

Instead of keeping the competition at bay by improving their own product, Lotus is suing those who make similar products, not on the grounds that the developers copied Lotus's software, but on the grounds that the competitor's products "look" too much like, and operate too much like, Lotus.

In a similar instance Apple sued Digital Research, claiming the DR's GEM looked too much like Apple's Macintosh software. The suit was resolved without any money changing hands. I suspect the agreement went something like this:

Look, if we can establish a precedent for making "Look and Feel" a copyrightable attribute, we can carve up the software market to suit ourselves. We can fend off the competition without having to compete. It won't matter if a competitor's product is better than ours, the fact that we used a particular "appearance" will be enough to bar newcomers from "our" market.

It seems to me that there are a number of very good Lotus- competitive products on the market right now: Supercalc 4, Borland's Quattro, and Microsoft's Excel to name just three.

If you are about to buy spreadsheet software for you business or home, I think you should consider whether or not you want to do business with a company that uses lawyers instead of excellence to keep itself ahead of its competitors.

If you feel that "Look and Feel" lawsuits are an undesireable business practice, consider boycotting companies using them to retard the progress so important to us all. That's the way we got rid of copy protection. It'll work in this instance, too.

REENTER not allowed now!

by Ralph Kenyon

Just recently I was editing a file and exited using ESC CTRL-X RETURN to find another *.ED file. Then I went to resume editing and got the dreaded "REENTER not allowed now!" message.

"Wait a minute!" I screamed. "I didn't do anything to deserve this!" I hadn't changed the disk, nor undeleted files, nor anything like that. So I tried "Get Edit" and "REENTER" ... No luck. Finally, I realized that my file was probably still in memory and I could get it back by EDiting a dummy file and using CTRL-U to get it back. Did you ever try this? It is slower than typing the file. Well, Rather than hold down the CTRL-U key, I figured that I might try defining a macro to do it for me. First I defined "1" as "^U^[1" and then I defind "2" as "^[1^[2" A macro that calls itself can only execute 16 times, so this way I got 16 x 16 undeletes or about 4 lines of text.

A half a page for each ESC 2 -- There's gotta be a faster way. So I redefined "1" as

"^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^U^[1"

This way it would be 16 times 16 times as many CTRL-U's as I just wildly typed. Then I pressed ESC 2 three more times and got back my whole file. By the way, this routine executes at full computer speed; it isn't slowed down by the keyboard repeat time limit.

Once no more characters would undelete,

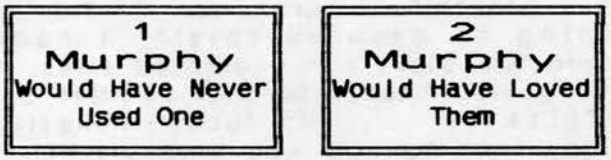
I found that I had two copies of my file, with some distortion in both copies. The way Edit stores the file is in two portions with the part before the cursor in low memory and the part after the cursor in high memory. Every time you press the cursor keys a character is moved from high to low (or vice versa) memory. But, Edit doesn't erase the spot where the character came from. That's why CTRL-U works at all; the characters are still there and delete just moves the pointer.

By moving the cursor from top to bottom and back you will have two complete copies (if your file is that small) of the file in memory. So, in recovering from this type of bomb the portion of the file up to where the cursor was will be in low memory and in perfect condition. The portion after the cursor will be in high memory and will likewise be in perfect condition. The rest is duplicative garbage and needs to be cleaned out. Of course, a very big file will have very little duplication in it.

[Of course, this won't work if the computer has been turned off, or another program (like BASIC) has been run which changes the contents of memory.]

Murphy's Computer Law

Reprinted from the Long Island Computer Association THE STACK, October, 1988.



BOVE'S THEOREM

The remaining work to finish in order to reach your goal increases as the deadline approaches.

BROOKS' LAW

Adding manpower to a late software project makes it later.

CANADA BILL JONES' MOTTO

It's morally wrong to allow naive end users to keep their money.

CANN'S AXIOM

When all else fails, read the instructions.

CLARKE'S THIRD LAW

Any sufficiently advanced technology is indistinguishable from magic.

DEADLINE-DAN'S DEMO DEMONSTRATION

The higher the "higher-ups" are who've come to see your demo, the lower your chances are of giving a successful one.

DEADLINE-DAN'S DEMON

Every task takes twice as long as you think it will take. If you double the time you think it will take, it will actually take four times as long.

DEMIAN'S OBSERVATION

There is always one item on the screen menu that is mislabeled and should read "ABANDON HOPE ALL YE WHO ENTER HERE".

DR. CALIGARI'S COME-BACK

A bad sector disk error occurs only after you've done several hours of work without performing a backup.

ESTRIDGE'S LAW

No matter how large and standardized the marketplace is, IBM can redefine it.

FINAGLE'S RULES:

1) To study an application best, understand it thoroughly before you start.
2) Always keep a record of data. It indicates you've been working.
3) Always draw your curves, then plot the reading.
4) In case of doubt, make it sound convincing.
5) Program results should always be reproducible. They should all fail in the same way.
6) Do not believe in miracles. Rely on them.

FRANKLIN'S RULE

Blessed is the end user who expects nothing, for he/she will not be disappointed.

GILB'S LAWS OF UNRELIABILITY

1) At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer.
2) Any system which depends on human reliability is unreliable.
3) Undetectable errors are infinite in variety, in contrast to detectable errors, which by definition are limited.
4) Investment in reliability will increase until it exceeds the probable cost of errors, or until someone insists on getting some useful work done.

GUMMIDGE'S LAW

The amount of expertise varies in inverse proportion to the number of statements understood by the general public.

HARP'S COROLLARY TO ESTRIDGE'S LAW

Your "IBM PC-compatible" computer grows more incompatible with every passing moment.

HELLER'S LAW

The first myth of management is that it exists.

HINDS' LAW OF COMPUTER PROGRAMMING

1) Any given program, when running, is obsolete.
2) If a program is useful, it will have to be changed.
3) If a program is useless, it will have to be documented.
4) Any given program will expand to fill all available memory.
5) The value of a program is proportional to the weight of its output.
6) Program complexity grows until it exceeds the capability of the programmer who must maintain it.
7) Make it possible for programmers to write programs in English, and you will find that programmers cannot write in English.

HOARE'S LAW OF LARGE PROGRAMS

Inside every large program is a small program struggling to get out.

THE LAST ONE'S LAW OF PROGRAM GENERATIONS

A program generator creates programs that are more "buggy" than the program generator.

MESKIMEN'S LAW

There's never time to do it right, but always time to do it over.

MURPHY'S FOURTH LAW

If there is a possibility of several things going wrong, the one that will cause the most damage will be the one to go wrong.

MURPHY'S LAW OF THERMODYNAMICS

Things get worse under pressure.

NINETY-NINETY RULE OF PROJECT SCHEDULES

The first ninety percent of the task takes ninety percent of the time, and the last ten percent takes the other ninety percent.

NIXON'S THEOREM

The man who can smile when things go wrong has thought of someone he can blame it on.

NOLAN'S PLACEBO

An ounce of image is worth a pound of performance.

OSBORN'S LAW

Variables won't, constants aren't.

O'TOOLE'S COMMENTARY ON MURPHY'S LAW

Murphy was an optimist.

PEER'S LAW

The solution to a problem changes the problem.

RHODES' COROLLARY TO HOARE'S LAW

Inside every complex and unworkable program is a useful routine struggling to be free.

ROBERT E. LEE'S TRUCE

Judgment comes from experience; experience comes from poor judgment.

SATTINGER'S LAW

It works better if you plug it in.

SHAW'S PRINCIPLE

Build a system that even a fool can use, and only a fool will want to use it.

SNAFU EQUATIONS

1) Given any problem containing N equations, there will be N + 1 unknowns.
2) An object or bit of information most needed will be least available.
3) Any device requiring service or adjustment will be least accessible.
4) Interchangeable devices won't.
5) In any human endeavor, once you have exhausted all possibilities and fail, there will be one solution, simple and obvious, highly visible to everyone else.
6) Badness comes in waves.

THOREAU'S THEORIES OF ADAPTATION

1) After months of training and you finally understand all of a program's commands, a revised version of the program arrives with an all-new command structure.
2) After designing a useful routine that gets around a familiar "bug" in the system, the system is revised, the "bug" is taken away, and you're left with a useless routine.
3) Efforts in improving a program's "user

friendliness" invariably lead to work in improving user's "computer literacy."
4) That's not a "bug", that's a feature!

WEINBERG'S COROLLARY

An expert is a person who avoids the small errors while sweeping on to the grand fallacy.

WEINBERG'S LAW

If builders built buildings the way programmers write programs, then the first woodpecker that came along would destroy civilization.

WOOD'S AXIOM

As soon as a still-to-be-finished computer task becomes a life- or-death situation, the power fails.

ZYMURGY'S FIRST LAW OF EVOLVING SYSTEM DYNAMICS

Once you open a can of worms, the only way to recan them is to use a larger can.

People

Long-time PolyLetter readers may be interested in a news item

Former PolyLetter editor Bob Bybee has recently left Chromatics, and accepted a position as Principal Hardware Engineer with Telecorp Systems of Norcross, Georgia. Telecorp Systems manufactures automated telephone dialing and answering equipment and was founded seven years ago by Mark Sutherland, the original publisher of PolyLetter.

Advertising

Commercial advertising rates are $50 for a full page, $25 for a half page, and $15 for a quarter page. Anything smaller is $3.00 per column inch. A column is 3-3/4 inches wide by 10 inches tall. A full page is 7-5/8 inches wide. Noncommercial ads by subscribers are free.

FOR SALE: 400 4116 memory chips. $200 for the lot. Charles Steinhauser, 2213 Voorhees, Apt. 102, Redondo Beach, CA 90278, Phone 1-213-214-0326.

FOR SALE: Poly 8810 box with power supply and mother board. $50 plus shipping. Charles A. Thompson, 2909 Rosedale Avenue, Dallas, Texas 75205-1532, Phone: (214)-368-8223

For Sale: 8813 Twin system with MS and 10M Priam HD, 8813 DSDD 5" with 10M Priam HD, 3 VTI boards, 1 SD controller, Poly Keyboard & Screen enclosure, 3 Serial cards, 1 Cassette interface, 1 Disk Controller Tester, 20 Boxes 8" disks, Field Service Manual, Dealer's kits, Manuals, etc. $1,500 or best offer for the lot. - Charles Trayser, 415-651-5931.

Abstract Systems, etc.
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

DISKS - MODEMS - PROMS - SOFTWARE - SPELL

1. MAXELL diskettes: 5-1/4" 10 hard sector -- $15 per box.
2. HayesSys modem software (for the Micromodem 100) $30.
3. [AiS] Spell, a good spelling checker for $35.
4. Abstract Systems Exec (Enhancements & bugs corrected) $35.
5. Abstract Systems Proms (Enhancements & bugs corrected) $35.
6. PolyGlot Library Volumes: $6 each.

(Send $1.00 for a complete catalog--[free with any order].)
(Make check or money order payable to Ralph Kenyon.)

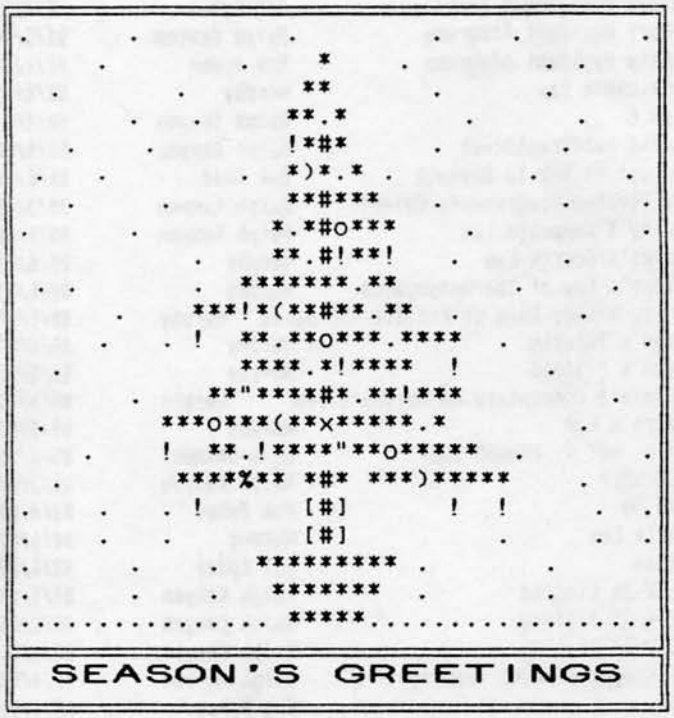
Readers Requests

Readers have asked for articles about the following: Assembly language article governing the use of WH0, WH1, Ckdr, Msg, etc. How would CP/M be of use. How does CP/M work. Where to get Drive Service, Keyboard Service, etc. Also wanted are hardware update recommendations, source lists, Communication software articles, File transfer to other computers. More on PClones. More articles on Hardware (Boards, etc.) More articles on languages. An explanation of relocatable files.

As time and space permits, I will try to answer all these questions. However, our readers are encouraged to submit their own articles on these and any subjects. Articles should be submitted on Poly 5" SSSD disk. PC disk format is also acceptable.

Annual Index

Bit Bucket



In This Issue

Coming Soon

Chip Logic families by Bob Bybee, Modems and Communications software; More: BASIC for Beginners, System Programmers Notes, Help, BugNotes, Public Domain Software, etc.

Questions

Can you find and answer the questions asked in this issue? Send your answers and requests in.

PolyLetter
191 White Oaks Road
Williamstown, MA 01267
(413) 458-3597

Address Correction Requested

FIRST CLASS MAIL

Ralph E. Kenyon, Jr. EXP: 9999
Abstract Systems, etc. 184
191 White Oaks Road
Williamstown, MA 01267-2256

PolyLetter Editor and Publisher: Ralph Kenyon. Subscriptions: US $18.00 yr., Canada $20.00 yr., Overseas $25.00 yr., payable in US dollars to Ralph Kenyon. Editorial Contributions: Your contributions to this newsletter are always welcome. Articles, suggestions, for articles, or questions you'd like answered are readily accepted. This is your newsletter; please help support it. Non-commercial subscriber ads are free of charge. PolyLetter is not affiliated with PolyMorphic Systems.

Back volumes of *PolyLetter* are available at the same price as the previous subscription rate. (US $15.00 yr., Canada $18.00 yr., Overseas $20.00 yr., payabh% in US dollars to Ralph Kenyon.) Individual issues are also available ($3.50, $4.00, $5.00).